

ISTQB® Software Testing Foundation

Course Book

Course Book

Contents

TIMETABLE	1
INTRODUCTION	5
SESSION 1: FUNDAMENTALS OF TESTING	8
Session 1: Fundamental of Testing - Notes	35
SESSION 2: TESTING THROUGHOUT THE SOFTWARE LIFECYCLE	53
Session 2: Testing Throughout the Software Lifecycle - Notes	75
SESSION 3: STATIC TECHNIQUES	93
Session 3: Static Techniques - Notes	107
SESSION 4: TEST DESIGN TECHNIQUES	118
Session 4: Test Design Techniques - Notes	163
SESSION 5: TEST MANAGEMENT	185
Session 5: Test Management - Notes	217
SESSION 6: TOOL SUPPORT FOR TESTING	237
Tool Summary	245
Session 6: Tool Support for Testing - Notes	254
EXERCISE	268
EXERCISE SOLUTION	307
PRACTICE EXAM	326
PRACTICE EXAM ANSWERS	328
MOCK EXAM	414
MOCK EXAM ANSWERS	420
APPENDIX A: SYLLABUS	483
APPENDIX B: GLOSSARY	539
APPENDIX C: RELEASE NOTES	597
APPENDIX D: PARTICIPANT FEEDBACK FORM	599

Timetable

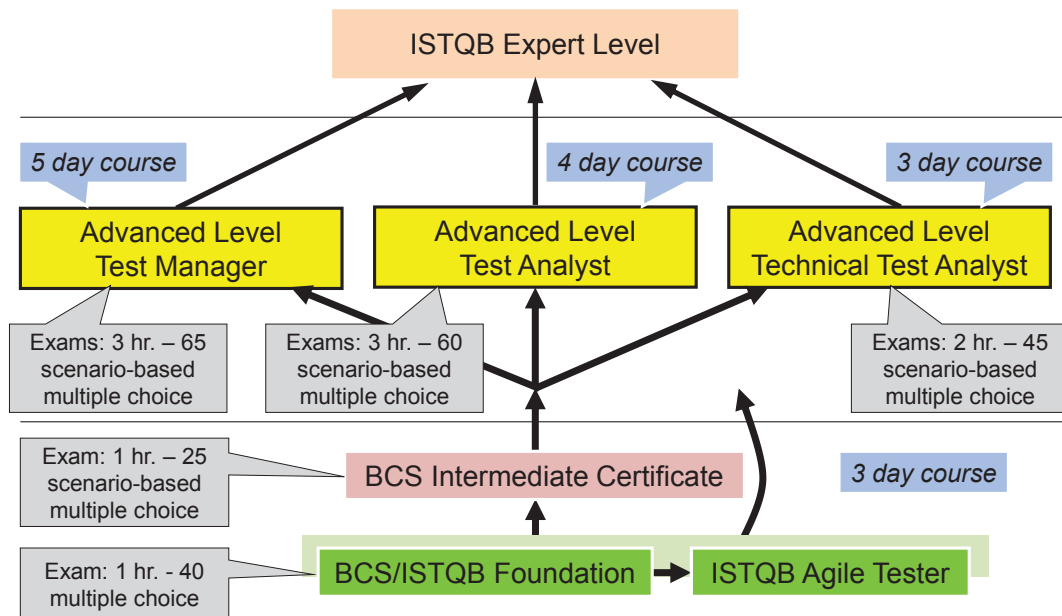
Start Time	Duration	Day 1 of 3
08:30	0:30	Welcome and Introductions
1. Fundamentals of Testing		
09:00	0:30	Why is testing necessary
09:30	0:10	Break
09:40	0:20	What is testing
10:00	0:35	General testing principles
10:35	0:20	Exercise 1_1: Match test activities to test process
10:55	0:10	Break
11:05	0:15	Fundamental test process
11:20	0:20	The psychology of testing
11:40	0:05	Code of ethics
11:45	0:05	Summary
2. Testing throughout the software lifecycle		
11:50	0:20	Software development models
12:10	0:20	Practice Exam 1 (including explanation of exams)
12:30	1:00	Lunch
13:30	0:10	Practice Exam 1 review
13:40	0:30	Test levels
14:10	0:10	Tool categories
14:20	0:10	Exercise 2_1: Test Levels
14:30	0:10	Break
14:40	0:10	Test types (the targets of testing)
14:50	0:05	Tool categories
14:55	0:10	Exercise 2_2: Test Targets
15:05	0:10	Maintenance testing
15:15	0:05	Summary
15:20	0:15	Learning Game: (Bingo Day 1) Terminology Review
3. Static techniques		
15:35	0:15	Static techniques and the test process
15:50	0:10	Break
16:00	0:25	The review process
16:25	0:05	Tool categories
16:30	0:10	Exercise 3_1: Reviews
16:40	0:20	Practice Exam 2
17:00		Close
	0:40	Homework: Sample Paper 1

Start Time	Duration	Day 2 of 3
08:30	0:10	Practice Exam 2 / Sample Paper 1 review
3. Static techniques, cont.		
08:40	0:17	Static analysis by tools
08:57	0:03	Tool categories
09:00	0:05	Summary
4. Test design techniques		
09:05	0:15	The test development process
09:20	0:05	Tool categories
09:25	0:05	Exercise 4_1: Black Box Exercise - Analyse the basis
09:30	0:10	Break
09:40	0:15	Categories of test design techniques
09:55	0:20	Specification-based: Equivalence partitioning, BVA
10:15	0:20	Exercise 4_2: EP / BVA Exercises
10:35	0:15	Specification-based: Decision tables
10:50	0:10	Break
11:00	0:15	Exercise 4_3: Decision tables
11:15	0:15	Specification-based: State transition testing
11:30	0:20	Exercise 4_4: State transition
11:50	0:05	Exercise 4_1: Black Box Exercise - Produce Test Procedures
11:55	0:15	Specification-based: Use case testing
12:10	0:15	Practice Exam 3
12:25	1:00	Lunch
13:25	0:10	Practice Exam 3 review
13:35	0:25	Structure-based: Statement and decision testing and coverage
14:00	0:30	Exercise 4_5: Statement and decision testing and coverage
14:30	0:10	Break
14:40	0:10	Other structure-based techniques
14:50	0:05	Tool categories
14:55	0:20	Experience-based techniques
15:15	0:15	Choosing test techniques
15:30	0:05	Summary
15:35	0:15	Learning Game: (Bingo Day 2) Terminology Review
15:50	0:10	Break
5. Test Management		
16:00	0:30	Test organisation
16:30	0:30	Practice Exam 4
17:00		Close
	1:00	Homework: Sample Paper 2

Start Time	Duration	Day 3 of 3
08:30	0:30	Practice Exam 4 / Sample Paper 2 review
5. Test Management, cont.		
09:00	0:50	Test planning and estimation
09:50	0:15	Test progress monitoring and control
10:05	0:05	Tool categories
10:10	0:10	Break
10:20	0:10	Configuration management
10:30	0:05	Tool categories
10:35	0:30	Risk and testing
11:05	0:15	Incident management
11:20	0:05	Tool categories
11:25	0:05	Summary
11:30	0:10	Break
11:40	0:25	Exercise 5_1: Incident report
6 Tool support for testing		
12:25	0:15	Introducing a tool into an organisation
12:40	0:05	Summary
12:45	0:20	Exercise 6_1: Tool Pairs Game
13:05	0:15	Practice Exam 5
13:20	1:00	Lunch
14:20	0:10	Practice Exam 5 review
14:30	1:30	Revision time / Sample Paper 3 - 30 Questions (30 Mins)
16:00	0:15	Break
16:15	1:00	Exam
17:15		Close

Introduction

BCS / ISTQB Professional Certificates



2

Learning Objectives (LO)

- the syllabus requires four levels of knowledge:
 - K1: remember, recall (e.g. recognise a definition)
 - K2: understand, give reasons for (e.g. why should testing start early in the life cycle)
 - K3: apply, do, perform (e.g. apply boundary value analysis technique to identify valid boundaries)
 - K4: analyse, solve a problem or task (e.g. analyse a document or code)
- K3: 4 black box techniques and how to write an incident report
- K4: 2 white box techniques

3

Course Content

1. Fundamentals of testing
 2. Testing throughout the software lifecycle
 3. Static testing techniques
 4. Test design techniques
 5. Test management
 6. Tool support for testing
- ISTQB exam

4

Course Notes

1. Fundamentals of testing
2. Testing throughout the software lifecycle
3. Static testing techniques
4. Test design techniques
5. Test management
6. Tool support for testing
7. Glossary
8. Syllabus

5

Session 1

Fundamentals of Testing

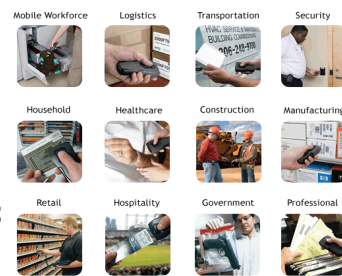
Contents

- 1.1 Why is Testing Necessary?
- 1.2 What is Testing?
- 1.3 Seven Testing Principles
- 1.4 The Fundamental Test Process
- 1.5 The Psychology of Testing
- 1.6 Code of Ethics

2

Software Systems Context

- where is software?
 - it has become an integral part of life
- does it always work correctly?
- defects in code, system & documents
- what can be affected?
 - company (loss of money, time, reputation)
 - environment (tank overflow, radiation leak)
 - people (medical devices)



How would your company be affected if you produced faulty software?

3

Effects of Defects - Examples

- Skype users
 - unable to login to accounts for nearly 2 days
- iPhone SMS messages
 - disable the alarm clock.
- Security breach of Apple's iPad
 - exposed personal information of AT&T customers, high-ranking government officials, celebrities and politicians.
- 8,500 people at a USA hospital thought they were still alive ...
 - but the hospital's computers were telling them they were not.

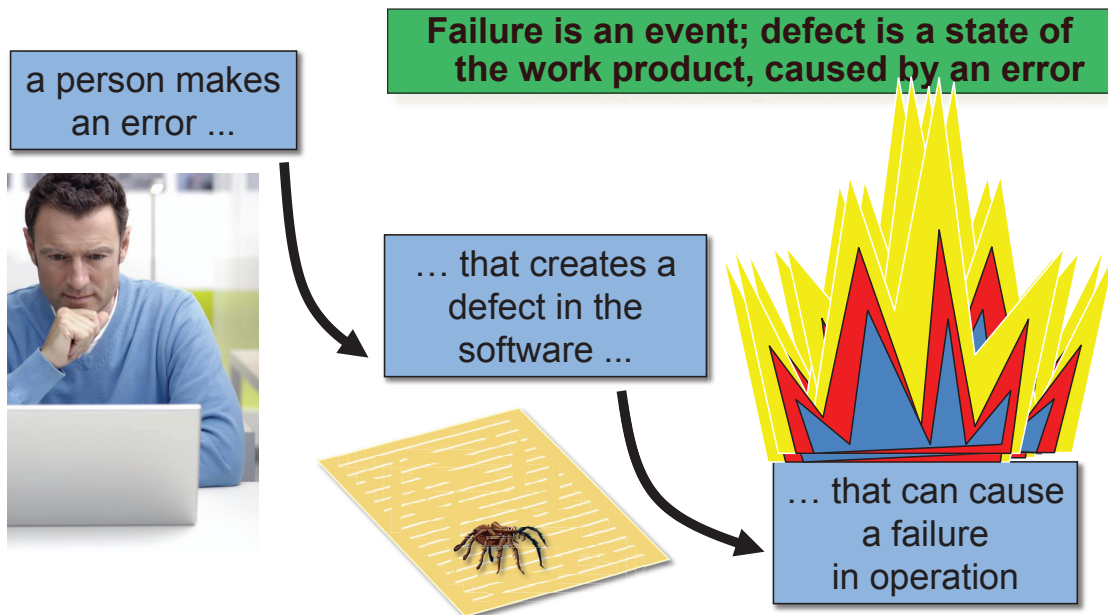
4

What is a “Bug”?

- **error** (mistake): a human action that produces an incorrect result
- **defect** (fault / bug): a flaw in code, software, system or document that can cause it to fail to perform its required function (e.g. incorrect statement or data definition)
 - if executed, a defect may cause a failure
- **failure**: actual deviation of the component or system from its expected delivery, service or result
 - can also be caused by an environment condition

5

Error - Defect - Failure



6

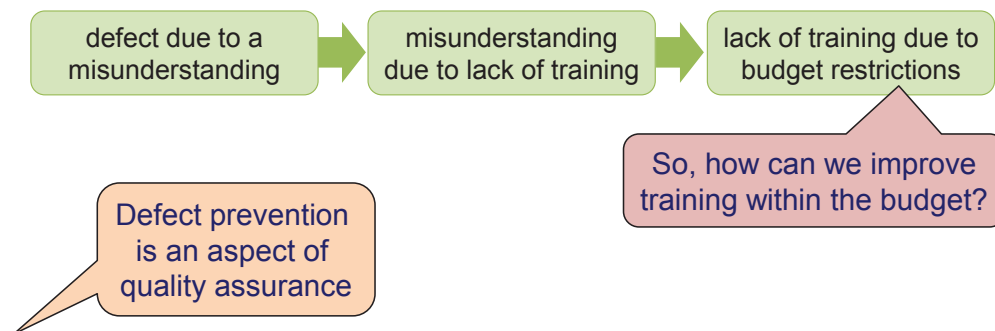
Causes of Defects

- people can make errors, errors can make defects
 - **human fallibility**
 - ▶ misunderstandings, wrong assumptions, time pressure
 - **complexity**
 - ▶ systems and infrastructure
 - **technology**
 - ▶ constantly changing, multiple interactions
 - **changing requirements**
 - ▶ changing business environments, need for competitive edge
- failures sometimes caused by physical environment
 - **radiation, magnetism, electronic fields, pollution**
 - ▶ can cause defects in firmware
 - ▶ or influence the execution of software
 - by changing hardware conditions

7

Root Causes

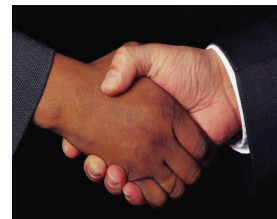
- root cause (a process issue)
 - “The source of a defect, such that if it is removed, the occurrence of the defect type is decreased or removed”
 - ▶ by understanding root causes, processes can be improved, thereby preventing future defects
- e.g.



8

The Role of Testing in Software Development, Maintenance and Operations

- in software development:
 - to find defects so that, when corrected
 - ▶ reduces risk of operational problems
 - ▶ contributes to increased quality
- in maintenance:
 - check changes and fixes do not adversely affect the working system
- in operations:
 - check conformance to expectations or SLA's
- in any phase:
 - to meet contractual or legal requirements, or industry-specific standards
 - ▶ e.g. medical, automotive, pharmaceutical



9

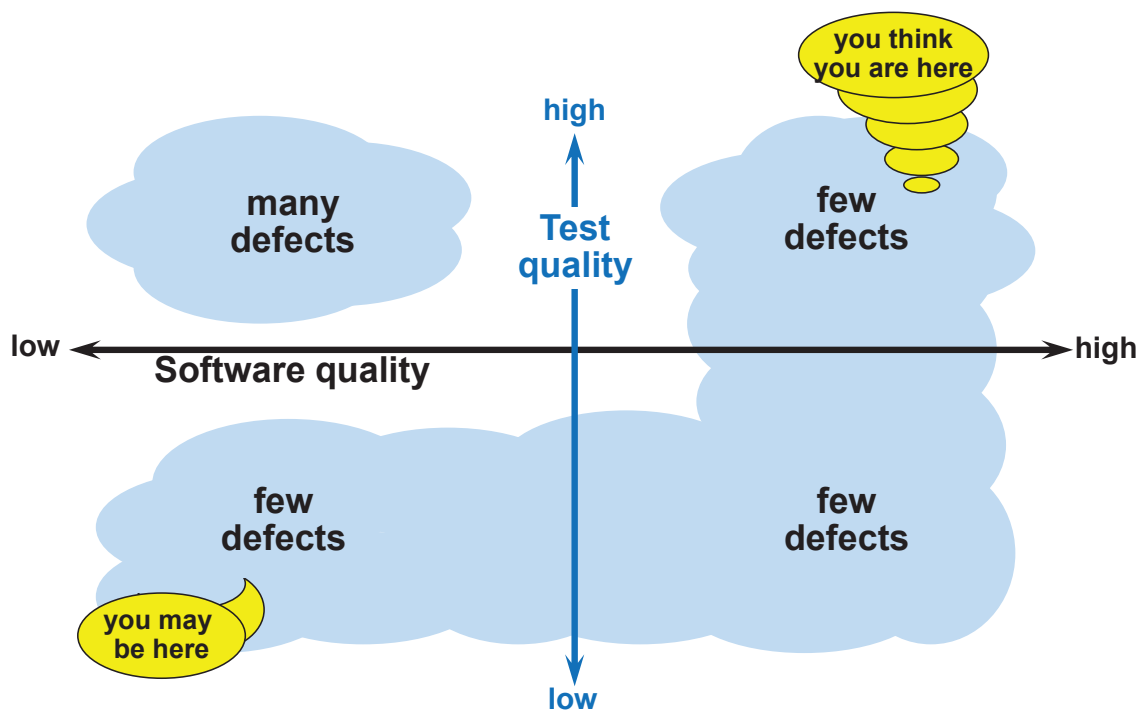
Testing and Quality

- testing can help measure software quality
 - number, type, severity and priority of the defects
 - ▶ both functional and non-functional characteristics
- testing can provide confidence in the software quality
 - if few or no defects are found (providing tests are good)
 - executed tests pass, i.e. meet the expected results
 - when risks are removed or reduced
- testing helps improve software quality
 - testing finds defects, development fixes the defects
- testing can help improve process quality
 - lessons learnt, root cause analysis and defect prevention

Testing is
part of
quality
assurance

10

Assessing Software Quality



11

How Much Testing is Enough?

- it depends on the risks for your system
 - technical and business risks (product / project)
 - ▶ expressed as likelihood times impact
 - risks are problems that may happen
- it depends on project constraints
 - e.g. time / budget
 - constraints are limitations that we have to meet
 - ▶ e.g. we have to deliver in 3 months

see session 5

Testing must provide enough information to stakeholders for them to make informed decisions on whether or not to release to the next stage.

12

So Why is Testing Necessary?

- to show that the software works **No**
- because software is likely to have defects **Yes**
- to learn about the reliability of the software **Yes**
- to fill the time between delivery of the software from development into testing and the release date **No**
- to prove that the software has no defects **No**
- because failures can be very expensive **Yes**
- because testing is included in the project plan **No**
- to avoid being sued by customers **Yes**
- to stay in business **Yes**

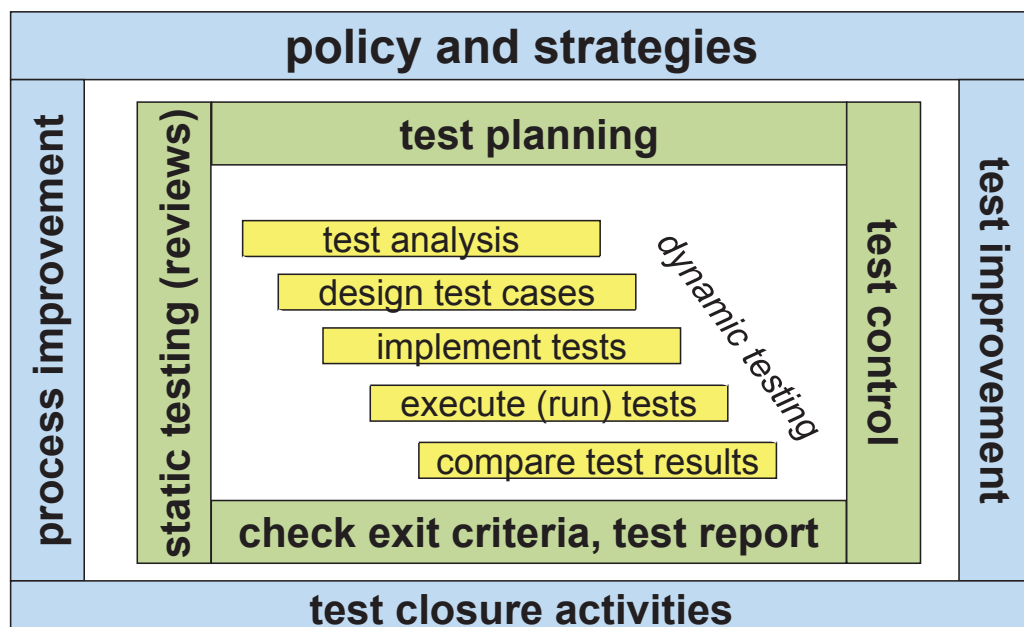
13

Contents

- 1.1 Why is Testing Necessary?
- 1.2 What is Testing?
- 1.3 Seven Testing Principles
- 1.4 The Fundamental Test Process
- 1.5 The Psychology of Testing
- 1.6 Code of Ethics

14

Testing is More than Execution



15

Static and Dynamic Testing - Differences

- static
 - software is not executed
 - ▶ doesn't run tests
 - trying to find defects
 - manual: reviews
 - ▶ of code / document
 - automated: static analysis
 - ▶ of code / document
 - ▶ finds all occurrences of detectable defects
- dynamic
 - software is executed
 - ▶ tests are run
 - trying to generate failures
 - manual or automated
 - often based on test cases
 - ▶ pre and post conditions, inputs, expected outcome
 - design of dynamic tests can help find defects early
 - ▶ sample of all possible tests

16

Objectives for Testing (Static and Dynamic)

- to generate failures (dynamic testing)
- to find defects (static testing)
- to gain confidence
 - about level of quality
- provide information for decision making
- to prevent future defects
 - reviews, early test design
- reduce or remove risks



17

Different Viewpoints of Testing Have Different Objectives

- testing during software development
 - to assess quality
 - ▶ report to stakeholders
 - static testing (reviews)
 - ▶ to find defects
 - so they can be fixed
 - dynamic testing (component, integration, system testing)
 - ▶ to cause failures
 - so defects can be identified and fixed
- acceptance testing
 - confirm the system works as expected, gain confidence
- maintenance testing
 - after changes, no new defects
- operational testing
 - assess system characteristics, e.g. performance, reliability



18

Testing and Debugging – Not the Same!

- testing
 1. generates failures (caused by defects)
 2. write incident report about it (if required)
 6. re-testing (to ensure a correct fix)
 7. regression test (to check for adverse impacts)
- done by
 - testers
 - developers (especially in component testing)
- debugging
 - “The process of finding, analysing and removing the causes of failures in software”

 3. identify the defect that caused the failure
 4. repair the defect in the code
 5. check that the defect is repaired correctly
 - done by
 - developers
 - ▶ whatever level of testing generated the failure

19

Contents

- 1.1 Why is Testing Necessary?
- 1.2 What is Testing?
- 1.3 Seven Testing Principles
- 1.4 The Fundamental Test Process
- 1.5 The Psychology of Testing
- 1.6 Code of Ethics

20

Seven Testing Principles

- principle:
 - fundamental truth
 - rule by which conduct may be guided
- testing principles:
 - suggested over the past 40+ years
 - offer general guidelines
 - common for all software testing
- ISTQB define 7 testing principles in the CTFL syllabus

21

Principle 1: Testing Shows Presence of Defects

- if all tests pass, is the software defect free? **No**
- if some tests fail, will we have found all the defects? **No**
- testing reduces the probability of undiscovered defects remaining
 - but not to 0%! 😞
 - no matter how much testing we do
 - ▶ we can't prove there are no defects
 - testing can prove there are defects
 - ▶ as soon as the first one is found

**1. Testing shows the presence of defects
but cannot prove that there are no defects**

22

Principle 2: Exhaustive Testing is Impossible

- what is exhaustive testing?
 - when all the testers are exhausted **No**
 - when all the planned tests have been executed **No**
 - exercising all combinations of inputs and preconditions **Yes**
- is exhaustive testing impossible?
 - it's not feasible (except for trivial cases)
 - we focus testing effort using risk and priorities

2. Exhaustive testing is impossible (in practice)

23

Why Not Just “Test Everything”?

- e.g. system inputs:
 - **type of loan**
 - ▶ 3 types
 - **amount of loan**
 - ▶ \$100 - \$10,000 in \$10 increments (1,000 values)
 - **duration of loan**
 - ▶ 1 to 120 months
 - **or monthly repayment**
 - ▶ \$100 - \$1,000 in \$10 increments (100 values)
- for exhaustive testing:
 - $3 * 1000 * 120 + 3 * 1000 * 100 = 660,000$ tests
 - at 1 second per test
 - 11,000 minutes (or 183 hours or 7.6 days)
 - ▶ not counting finger trouble, defects or re-testing
 - at 10 seconds per test
 - 11 weeks
 - at 1 minute per test
 - 15 months

24

Principle 3: Early Testing

- does it matter when the testing starts? **Yes**
 - defects found earlier are cheaper to fix
- which of the following should testing focus on?
 - upsetting developers? **No**
 - showing the software working? **No**
 - finding the showstoppers as quickly as possible? **Yes**

What test objectives do you have?

3. Test activities shall start as early as possible in the lifecycle and should focus on defined objectives

25

Principle 4: Defect Clustering

- true or false? (commonly)
 - defects are distributed evenly in the system **False**
 - a small number of modules:
 - ▶ contain most of the defects discovered in pre-release testing **True**
 - ▶ show the most operational failures **True**
 - testing should be more thorough where it is most needed **True**

4. A small number of modules usually contain most of the discovered defects or cause most of the operational failures.

- therefore testing effort should be focused proportionally to the expected and later observed module defect density

26

Principle 5: Pesticide Paradox

- true or false?
 - tests should find defects **True**
 - tests should be repeatable **True**
 - run the same test again - find a new defect? **Unlikely**
- how can we find a stronger pesticide (tests)?
 - regularly review and revise tests
 - write new and different tests to exercise different parts and potentially find more defects
 - run existing tests in a different order

5. Pesticide paradox: software becomes immune to tests

27

What is a Successful Test?



28

Principle 6: Testing is Context Dependent

- should all software be tested equally well? **No**
- is good testing consistent across different organisations? **No**
- testing should not be the same in every situation, e.g.
 - **safety-critical applications**
 - ▶ more formal, more thorough
 - **e-commerce**
 - ▶ careful balance between time, cost and quality
 - **new market, innovative product**
 - ▶ less formal, time to market more important



6 . Testing is context dependent

29

Principle 7: Absence-of-Errors Fallacy

- should every defect found by testing be fixed? **No**
- is it OK to release software with known defects in it? **Maybe**
- “Absence-of-errors fallacy”
 - suppose all defects found by testing are fixed – is the system now ok?
 - ▶ what if performance requirements were not specified?
 - ▶ users could find system unacceptably slow
 - they may not use a system even if it conforms to the requirements
 - fixing “defects” may not be the most important thing for users

7 . Absence of defects does not mean the system is ok

30

Question: Distribution of Defects

Which of the following is true?

The more often a test is run,

a) the more defects will be found

b) the less likely it is to find a new defect

c) the more likely it is to find a new defect

d) the better that test will become



31

Question: Amount of Testing

Which of the following is true?

- a) good testers try to implement the same practices whatever organisation they work in
- b) it's okay if testing is less thorough in some organisations**
- c) organisations that don't test thoroughly are not doing a good job
- d) good testing is consistent across different organisations



32

Question: Importance of Finding and Fixing Defects

Which of the following is true?

- a) finding defects always helps the users
- b) any defect fixed is helpful to the users
- c) no defects means high quality for users
- d) minimising the number of defects is **NOT** the most important thing for users**



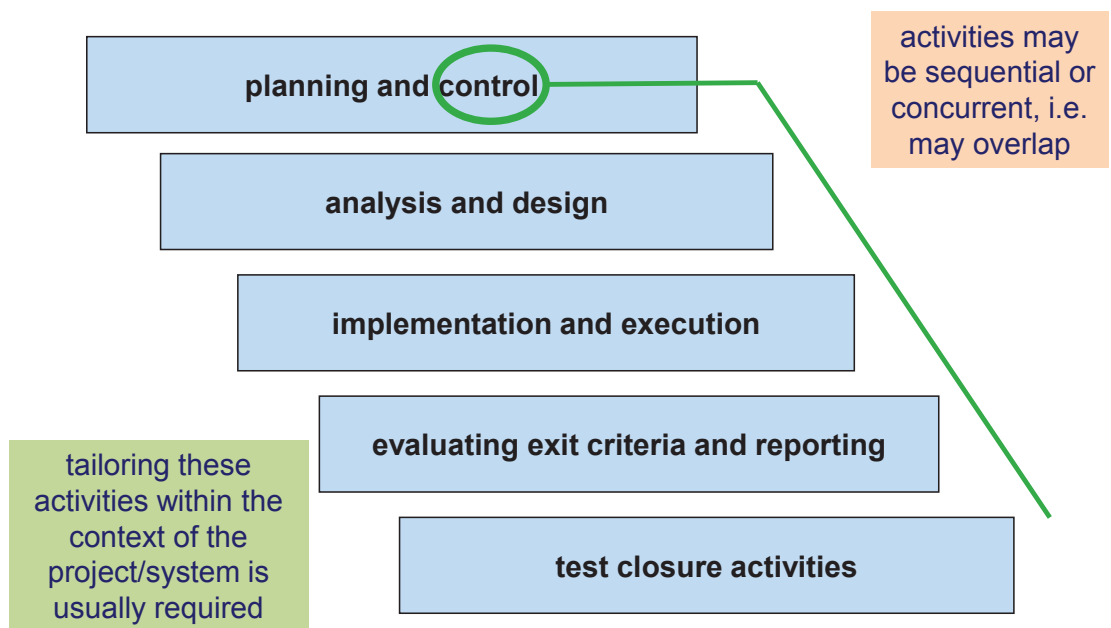
33

Contents

- 1.1 Why is Testing Necessary?
- 1.2 What is Testing?
- 1.3 Seven Testing Principles
- 1.4 The Fundamental Test Process**
- 1.5 The Psychology of Testing
- 1.6 Code of Ethics

34

The Fundamental Test Process

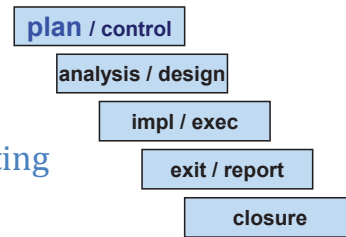


35

Test Planning

- tasks

- implement test policy / strategy
- determine scope, risks, and objectives of testing
- determine the test approach (strategy)
 - ▶ test levels, entry/exit criteria, techniques, automation
- integrating test activities into software life cycle
- deciding what to test, roles
- assign the test resources (people, environment)
- schedule test activities (design/build/exec tests, etc.)
- defining test documentation requirements
- selecting metrics for monitoring and control

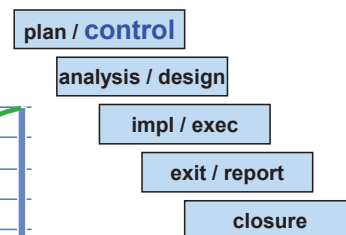
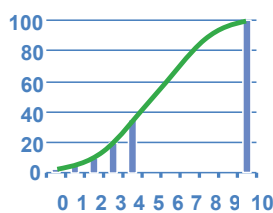


36

Test Control (Continuous)

- tasks

- measure and analyse results
- monitor and document
 - ▶ progress, test coverage
- take actions
 - ▶ mostly corrective
 - e.g. reprioritise tests when a new risk is identified
 - e.g. set an entry criterion for fixes (such as re-testing by a developer)
 - ▶ can be pre-emptive
 - e.g. change test schedule for environment availability
- make decisions (who, what, when, where, etc.)



37

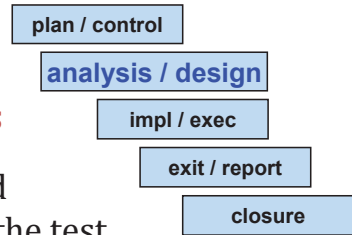
Test Analysis and Design

analysis tasks

- review the test basis
 - requirements, architecture, design, interfaces, etc.
- evaluate testability of the test basis & test objects
- identify and prioritise test conditions
 - based on test items, test basis, software behaviour and structure

design tasks

- design and prioritise the test cases
- identify necessary test data (supporting test cases)
- design test environment, identify required infrastructure & tools
- create bi-directional traceability between test basis and test cases



38

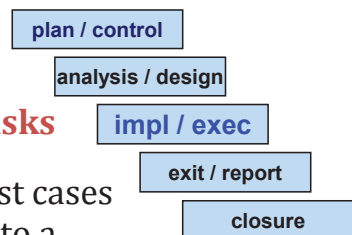
Test Implementation and Execution

implementation tasks

- implement and prioritise test cases, identify test data
- develop test procedures, create test data
- prepare test harnesses, write automated test scripts
- create efficient test suites
- verify test environment
- verify bi-directional traceability
 - test basis to test cases

execution tasks

- execute test cases according to a planned sequence
- log outcomes, record SUT, tool, and testware versions
- compare actual / expected
- report discrepancies as incidents, analysing cause
 - defect in code, test data / document, procedure
- repeat activities as needed
 - re-testing, regression



39

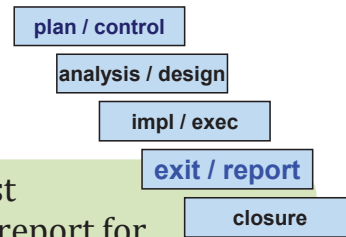
Evaluating Exit Criteria and Reporting

evaluating exit criteria

- assess test execution against the defined test objectives
 - for each test level
- tasks
 - check test logs against exit criteria specified in test plan
 - assess if more tests are needed
 - ▶ or if exit criteria should be changed

reporting

- write a test summary report for stakeholders
 - used when making a release decision
- tasks
 - summarise what happened during testing
 - give an evaluation of the software quality

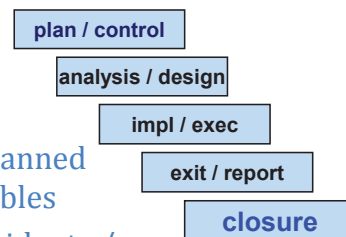


40

Test Closure

- occurs at key project milestones
 - e.g. end of a test level, system released, test project completed, milestone achieved, maintenance release completed
- consolidate experience, testware, facts
 - based on data from completed activities

- tasks
 - check planned deliverables
 - close incidents / raise changes
 - doc. system acceptance
 - archive testware, environment and infrastructure for reuse
 - handover of testware to maintenance organisation
 - analyse lessons learned, improve test maturity



41

Contents

- 1.1 Why is Testing Necessary?
- 1.2 What is Testing?
- 1.3 Seven Testing Principles
- 1.4 The Fundamental Test Process
- 1.5 The Psychology of Testing
- 1.6 Code of Ethics

42

Developer / Tester Mindset

- | | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">● developer needs to<ul style="list-style-type: none">■ analyse requirements■ design technical solution to implement detailed requirements■ solve problems in implementation● how can I make the computer do what I want it to? (make this work) | <ul style="list-style-type: none">● tester needs to<ul style="list-style-type: none">■ analyse requirements■ identify test conditions■ design test cases, test data● what could go wrong?● what if the assumption does not hold?● how could it break?● how does it interact with the rest of the system? |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Can / should developers test their own work?

43

Independence

- test your own work?
 - likely to find only 30% to 50% of defects (*source: Open University*)
 - why?
 - same assumptions / thought processes
 - see what you meant, not what is there
 - emotional attachment
 - ▶ no motivation to find defects
 - ▶ high motivation not to see defects
 - even if they are there!
 - levels of independence*
 - developers test their own code
 - developers test each other's code
 - tester on the development team
 - independent test team
 - testers from business / users
 - test specialists
 - ▶ e.g. usability or performance
 - independent testers
 - ▶ e.g. outsourced, external to organisation
- * revisited in session 5

44

Aspects of Independence

- people are driven by objectives
 - “find defects” versus “confirm the software works”
 - clear objectives are very important!
- independent testing can be done at any level
 - e.g. component, integration, system, acceptance
 - reviews are a form of independent testing too
- independence does not replace familiarity
 - yes, developers can test their own work
 - but independent testers may find more or different defects

45

Who Wants to be a Tester?

- a destructive process
- under worst time pressure (at the end)
- bring bad news (“your baby is ugly”)
- how should defect information be communicated (to authors and managers?)
 - constructively, correctly, objectively, fact-based, neutral
 - focus on common goal of better quality systems
 - confirm that both sides have understood correctly
 - understand how the other person feels



46

Who Wants to be a Developer?

- spent a lot of time and finally got this to work
- a constructive activity, feeling of accomplishment
- hearing what the tester says:
 - a destructive activity - on my code
 - ▶ working against me
 - “your product has defects”
 - ▶ you aren’t very good at your job
 - reaction?
 - ▶ defensiveness, denial, anger, frustration

47

Is Testing Destructive or Constructive?

- testing is destructive
 - “breaks” the software
 - ▶ like a toy’s safety test
- testing is constructive
 - helps to manage risks, reducing product risks
 - defect information can help improve development and test processes
 - defects found early will save time and money later



48

The Psychology of Successful Testing

- key aspects
 - clear objectives (understood)
 - balance between self-testing and independent testing
 - courteous factual communication
 - collaborate, not battles – get on the same side

49

Characteristics of a Good Tester

- curiosity, professional pessimism, critical eye
- attention to detail
- good communication skills
- experience (for error guessing)

50

Question: Fundamental Test Process (1)

Producing a 'step by step' set of instructions (test procedure) is part of which stage of the fundamental test process?



a) execution

b) analysis

c) design

d) implementation

51

Question: Fundamental Test Process (2)

Assessing whether more tests are needed is a task done at which stage of the fundamental test process?



- a) analysis and design
- b) implementation and execution
- c) evaluating exit criteria and reporting
- d) test closure

52

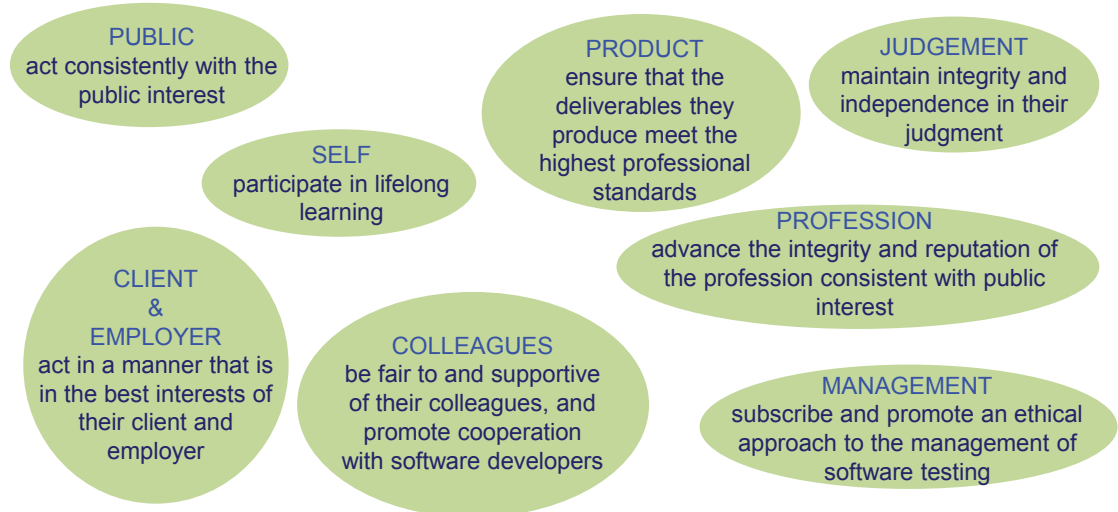
Contents

- 1.1 Why is Testing Necessary?
- 1.2 What is Testing?
- 1.3 Seven Testing Principles
- 1.4 The Fundamental Test Process
- 1.5 The Psychology of Testing
- 1.6 Code of Ethics

53

The Tester's Code of Ethics

- certified software testers shall...



54

Summary – Key Points

- testing is necessary because: failures can be expensive
- causes of error: human fallibility, time pressure, complexity
- objectives of testing: find defects, build confidence, provide information, prevent defects
- testing principles: presence, exhaustive, early, clustering, pesticide paradox, context dependent, absence-of-defects
- the test process: planning & control, analysis & design, implementation & execution, evaluating exit criteria & reporting, closure
- aspects of successful testing: clear objectives, independence, good communication

55

SESSION 1: FUNDAMENTAL OF TESTING - NOTES

Terms

Bug, confirmation testing, debugging, defect, error, error guessing, exhaustive testing, exit criteria, failure, fault, incident, independence of testing, mistake, quality, regression testing, requirement, re-testing, review, risk, test case, testing, test basis, test condition, test coverage, test data, test execution, test log, test objective, test plan, test procedure, test policy, test suite, test summary report, testware.

From the ISTQB Glossary

bug: See defect.

confirmation testing: Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

debugging: The process of finding, analysing and removing the causes of failures in software.

defect: A flaw in a component or system that can cause the component or system to fail to perform its required function, e.g., an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system.

error: A human action that produces an incorrect result.

error guessing: A test design technique where the experience of the tester is used to anticipate what defects might be present in the component or system under test as a result of errors made, and to design tests specifically to expose them.

exhaustive testing: A test approach in which the test suite comprises all combinations of input values and preconditions.

exit criteria: The set of generic and specific conditions, agreed upon with the stakeholders for permitting a process to be officially completed. The purpose of exit criteria is to prevent a task from being considered completed when there are still outstanding parts of the task which have not been finished. Exit criteria are used to report against and to plan when to stop testing.

failure: Deviation of the component or system from its expected delivery, service or result.

fault: See defect.

incident: Any event occurring that requires investigation.

independence of testing: Separation of responsibilities, which encourages the accomplishment of objective testing.

mistake: See error.

quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

regression testing: Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

requirement: A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

re-testing: See confirmation testing.

review: An evaluation of a product or project status to ascertain discrepancies from planned results and to recommend improvements. Examples include management review, informal review, technical review, inspection, and walkthrough.

risk: A factor that could result in future negative consequences.

test case: A set of input values, execution preconditions, expected results and execution post-conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

testing: The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

test basis: All documents from which the requirements of a component or system can be inferred. The documentation on which the test cases are based. If a document can be amended only by way of formal amendment procedure, then the test basis is called a frozen test basis.

test condition: An item or event of a component or system that could be verified by one or more test cases, e.g., a function, transaction, feature, quality attribute, or structural element.

test coverage: The degree, expressed as a percentage, to which a specified coverage item has been exercised by a test suite.

test data: Data that exists (for example, in a database) before a test is executed, and that affects or is affected by the component or system under test.

test execution: The process of running a test on the component or system under test, producing actual result(s).

test log: A chronological record of relevant details about the execution of tests.

test objective: A reason or purpose for designing and executing a test.

test plan: A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.

test policy: A high-level document describing the principles, approach and major objectives of the organization regarding testing.

test procedure: A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script.

test suite: A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one.

test summary report: A document summarizing testing activities and results. It also contains an evaluation of the corresponding test items against exit criteria.

testware: Artefacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.

1.1 Why is Testing Necessary?

Learning Objectives

LO-1.1.1	K2	Describe with examples the way in which a defect in software can cause harm to a person, the environment or to a company.
LO-1.1.2	K2	Distinguish between the root cause of a defect and its effects.
LO-1.1.3	K2	Give reasons why testing is necessary by giving examples.
LO-1.1.4	K2	Describe why testing is necessary by giving examples.
LO-1.1.5	K2	Explain and compare the terms error, defect, fault, failure, and the corresponding terms mistake and bug, using examples.

Terms

Bug, defect, error, failure, fault, mistake, quality, risk.

1.1.1 Software Systems Context

Software seems to be everywhere these days – not just processing your bank account details, and booking a holiday, but also in automotive systems (cruise control, braking systems), medical devices (pacemakers and insulin pumps), and even everyday appliances such as washing machines, burglar alarms, televisions, and clocks.

So does this software always work correctly? We tend to assume that it does, but is this always the case? We are used to finding problems in the software we use at work, so why should software embedded in appliances be much different?

When software doesn't work, this can have different effects from life threatening to being fairly insignificant.

These examples all come from the Internet:

- Skype users were unable to login to accounts for nearly 2 days because of a algorithmic software flaw Source: By Dr. Dobb's Journal, August 17, 2007
- iPhone had an incompatibility between the SMS and Alarm clock. If a user set the alarm for the morning and overnight received a text message the alarm would not go off. When the user looks at the phone there is a pop up message to say text message received. Once acknowledged by the user, the alarm will then go off! Source: www.stickyminds.com
- Security breach of Apple's iPad exposed personal information of AT&T customers, including those high-ranking government officials, celebrities and politicians. The FBI is currently looking into this. Source: BBC News Adobe acknowledges critical security flaw in software Dated Monday, 7 June 2010 16:35 UK
- 8500 people at a USA hospital thought they were still alive. But the hospital's computers were telling them they were not. A "mapping error" in the conversion process resulted in the hospital assigning a disposition code of "20" - which meant died - instead of "01," which meant the patient had been discharged. Worse, that errant data wasn't sent just to the shocked patients but to their insurance companies as well as the local Social Security office, which helps determine whether elderly or disabled patients are eligible for Medicare. Obviously, once a patient is dead, Medicare - assuming its electronic-records system is accurate - isn't going to make any payments on bills for future medical services or medication. St. Mary's Mercy officials scrambled to set up a hotline for affected patients, notified insurance companies and government agencies about the mistake and then went about fixing the code issue that caused the misidentification in the first place. Source: www.baselinemag.com

Testing is necessary because software is likely to have defects in it and it is better (cheaper, quicker and more expedient) to find and remove these defects before it is put into live operation. Failures that occur during live operation are much more expensive to deal with

than failures than occur during testing prior to the release of the software. Of course other consequences of a system failing during live operation include the possibility of the software supplier being sued by the customers!

Testing is essential in the development of any software application. Testing is needed in order to assess what the application actually does, and how well it does it, in its final environment. Developing an application without testing is merely a paper exercise - it may work or it may not, but there is no way of knowing it without testing it.

There are many examples where software fails and are being reported in the press all the time. If you look at the Stickyminds website there are many up to date examples to read about. The ones given on the slide are some current examples.

1.1.2 Causes of Software Defects

Three terms that have specific meanings are error, defect and failure.

Error or mistake: a human action that produces an incorrect result.

Defect, fault or bug: a flaw in code, software, system or document that can cause it to fail to perform its required function (e.g. incorrect statement or data definition).

Failure: actual deviation of the component or system from its expected delivery, service or result.

An error is something that a person does unintentionally. We all make mistakes, it is inevitable, especially when dealing with something as complex as a computer system.

The result of an error being made while developing software or writing a document is called a defect (also known as a fault or a bug). It is something that is wrong in the program code, or in the documentation (specifications, manuals, etc.).

If a defect is in program code and that code is executed, then the system may produce an incorrect result or not perform the correct action, this is known as a failure. Failures may be caused by defects in the software, but a software application can contain defects that never cause it to fail (for example, this can occur if the defects are in those parts of the application that are never used).

So a failure is something that happens, an event. It is visible in the outside world. A defect is something that is in some artefact – either the code itself or some document associated with a system. The defect is put into the program code or document because someone made an error (a mistake).

So why do people make these mistakes? One thing is sure – people don't put the defects into software products on purpose, yet the defects do exist. Mistakes are often due to the increasing complexity of the system and its interfaces, the fact that technology is constantly changing, and increasingly aggressive deadlines leading to excessive time pressure. Under these conditions, it is no wonder that mistakes can be made under pressure – this is just human nature and a fact of life.

Sometimes failures are caused by aspects of the physical environment, such as magnetism or radiation. For example, a computer sited next to a lift shaft failed when the lift went by (due to the effect of the electromagnetism created by the lift motors on the computer hardware). This was not a software problem, so the failure was not caused by a software defect!

1.1.3 The Role of Testing in Software Development, Maintenance and Operations

Testing is important in software development. Developers should test their own software as they develop it, or sometimes they write the tests before writing the code – this is known as test-driven development. By writing tests first, the code that is written to pass those tests tends to have fewer defects. Any defects that are found by running tests during development

can be corrected immediately, so fewer defects are released, i.e. we end up with better quality software.

Testing is important in maintenance, to help ensure as much as possible that enhancements подобрувања and defect fixes do not adversely негативно affect the working system.

Testing is important in operations, i.e. when the system is performing its intended function, to make sure as much as possible that the system continues to function correctly in real use.

Testing may also be required for legal or contractual reasons, and some industry sectors have their own standards for testing, e.g. airborne systems or pharmaceutical software.

1.1.4 Testing and Quality

Testing and quality go hand in hand! Basically we don't know how good the software is until we have run some tests. Once we have run some good tests we can state да се наведе how many defects we have found (of each severity level) and may also be able to predict how many defects remain (of each severity level).

Quality can be measured in terms of the defects found - the more defects found, the poorer the quality of the system being tested.

But what if no defects (or very few) are found? Does this mean that the system is of good quality? Not necessarily. It may well mean that the testing is very poor. (The best way to find no defects at all is to do no testing!) If the testing is known to be of good quality, then if few or no defects are found in testing we can be more justifiably confident that the system quality is good.

Testing can also help to increase зголеми the quality of a system, provided that the defects that are found are fixed.

Testing is used to measure the quality of a software system, both in terms of what it does (its functionality) and how well it does it (non-functional quality characteristics or attributes). The standard ISO 9126 ("Software Engineering – Software Product Quality") gives guidance on specifying, measuring and testing quality characteristics. There are also draft нацрт standards for non-functional testing that can be freely accessed on www.testingstandards.co.uk. Non-functional testing includes performance testing, reliability testing, usability testing and others. These are covered in more detail in Section 2.

Testing can also help to reduce risk. If certain areas of the system are critical to the stakeholders, more testing in that area will give more information about the quality of those areas and therefore will reduce the risk of unpleasant surprises in that area.

Testing can help to increase the quality of future systems, through highlighting lessons learned. At the end of a project, a post-project review meeting may be held to assess what went well and what could have been done better in that project. Understanding the root causes of defects, i.e. why someone made a particular error, can help to identify process improvements that will prevent that type of defect in the future (or at least make it less likely). This means that future systems can be of improved quality, not just this one.

How do we measure the quality of software? The number of defects found is a common way to start. The more defects we find, the worse the quality is, so we can easily measure poor quality software.

What about the converse: the fewer defects we find, the better the quality of the software? This is not true, since finding few or no defects can mean one of three things: good software, poor testing or both poor testing and poor software. Without knowing independently about the quality of the testing, no justified conclusions can be drawn about the quality of the software.

If you have a faulty measuring instrument, you cannot draw any justified conclusions about what you are measuring. If I discovered that I had lost weight, I might be delighted, but if my scales are broken, I am not actually any thinner - my delight is unjustified неоправдано.

If we do lots of testing, our confidence will rise, but this is because confidence is a psychological factor. Consider how differently you might regard a piece of software if you run

all of the easy tests first, so most of the early tests work correctly, compared to running all of the most difficult tests first, so that most of the early tests fail. The same software with the same set of tests would give two very different initial impressions to your assessment of confidence, even *дури и though иако* the quality of that software is the same each time.

You cannot have justified confidence in the quality of the software unless you have confidence in the quality of the testing.

A distinction can be made between the cause of a defect and the root cause. The **cause** is the immediate reason why someone made an error, for example, they misunderstood something, or they were careless and didn't double-check something. The **root cause** is an organisational issue *проблем* which makes it more likely that an employee will make this particular mistake. For example, they misunderstood because they had never been given the right background information or training. They may have been careless because of excessive deadline pressure, or because no one communicated to them the importance of double-checking this particular item.

Testing should be integrated into a full quality approach, normally as a quality assurance activity, along with development standards, defect analysis, training, etc.

1.1.5 How Much Testing is Enough?

This is a question to which there is not a clear-cut answer. This is a question where the answer is a decision that you make, based on a number of factors. The most important factors are those based on risk (which is covered in more detail in Session 5). The risk can be to a product (e.g. there is a risk that a technically complex area may have a defect), or may be a project risk (e.g. the software may not be delivered into System Test on time). Risks may be more technical in nature, or may be related to the business. For example, the system may not conform to the business process used by some users (product risk) or the key user acceptance tester may not be available when needed (project risk).

Project constraints *ограничувања* such as time or budget are also factors that help to determine how much testing is deemed to be enough. Although the deadline should not be the only factor that determines when testing is stopped, we can't just go on testing forever either.

Testing should provide information to the stakeholders of the system, so that they can make an informed decision about whether to release a system into production, or to customers. Testers are not responsible for making that decision – they are responsible for providing the information (the “project intelligence” as Paul Gerrard refers to it) so that the decision can be made in the light of good information.

It is possible to do *enough* testing but determining the how much is enough is difficult. Simply doing what is planned is not sufficient since it leaves the question as to how much should be planned. What is enough testing can only be confirmed by assessing the results of testing. If lots of defects are found with a set of planned tests it is likely that more tests will be required to assure that the required level of software quality is achieved *постигне*. On the other hand, if very few defects are found with the planned set of tests, then (providing the planned tests can be confirmed as being of good quality) no more tests will be required.

Saying that enough testing is done when the customers or end-users are happy is a bit late, even though it is a good measure of the success of testing. However, this may not be the best test stopping criteria to use if you have very demanding end-users who are never happy!

Why not stop testing when you have proved that the system works? It is *not possible to prove* that a system works without exhaustive testing (which is totally impractical for real systems).

Have you tested enough when you are confident that the system works correctly? This may be a reasonable test stopping criterion, but we need to understand how well justified that confidence is. It is easy to give yourself false confidence in the quality of a system if you do not do good testing.

Ultimately, the answer to "How much testing is enough?" is "It depends!" (this was first pointed out by Bill Hetzel in his book "The Complete Guide to Software Testing"). It depends on risk, the risk of missing defects, of incurring high failure costs, of losing credibility and market share. All of these suggest that more testing is better. However, it also depends on the risk of missing a market window and the risk of over-testing (doing ineffective testing) which suggest that less testing may be better.

We should use risk to determine where to place the emphasis when testing by prioritising our test cases. Different criteria can be used to prioritise testing including complexity, criticality, visibility and reliability.

We cannot test everything; it is easy to identify far more test cases than we will ever have time to execute so we need an approach to selecting a subset of them. Selecting test cases at random is not an effective strategy. We need to use a more intelligent approach that helps identify which tests are most important. In short, we must prioritise our tests, as will be covered in more detail in Session 5.

There are many different criteria that can be used to prioritise tests and they can be used in combination. Possible ranking criteria include the following:

- tests that would find the most severe *тешки* failures;
- tests that would find the most visible failures;
- tests that would find the most likely defects;
- ask the end-user to prioritise the requirements (and test those first);
- test first the areas of the software that have had most defects in the past;
- test most those areas of the software that are most complex or critical.

There is further discussion and examples of how to prioritise tests in sessions 4 and 5.

Other factors that affect our decision on how much testing to perform include possible contractual obligations. For example, a contract between a customer and a software supplier for a bespoke system may require the supplier to achieve 100% statement coverage (coverage measures are discussed in session 3). Similarly, legal requirements may impose a particular degree of thoroughness in testing although it is more likely that any legal requirements will require detailed records to be kept (this could add to the administration costs of the testing).

In some industries (such as the pharmaceutical industry and safety critical industries such as railroad switching and air traffic control) there are standards defined that have the intent of ensuring rigorous testing.

1.2 What is Testing?

Learning Objectives:

- | | | |
|----------|----|------------------------------------------------------------------------------------------------|
| LO-1.2.1 | K1 | Recall <i>потсетиме</i> the common <i>заеднички/чести</i> objectives of testing. |
| LO-1.2.2 | K2 | Provide examples for the objectives of testing in different phases of the software life cycle. |
| LO-1.2.3 | K2 | Differentiate testing from debugging. |
-

Terms

Debugging, requirement, review, test case, testing, test objective.

“Testing is more than testing”

Software testing is much more than just executing tests and checking the results. The test process starts much earlier in the project with activities such as test planning and reviewing documents (particularly specifications and source code). Other test activities occur before test execution including identifying test conditions, designing test cases, and implementing test procedures. Many test activities occur after test execution including reporting test failures, reporting test progress, and evaluating exit criteria.

In common with executing tests most of the activities mentioned above are categorised as a dynamic testing activities because they are associated with executing (running) tests on the software. There are also static testing activities including reviewing documents (as mentioned above) and static analysis (a tool supported form of static testing). (See section 0 фаги for a discussion on static and dynamic testing).

There are still further activities that surround both static and dynamic testing activities. Test policies and strategies help to determine the approach to planning the testing. At the end of a project, the test closure activities will look back and put in place process improvements, including test improvements, which can impact on future strategies.

Objectives of Testing

There can be different objectives for testing, depending on the stage in the lifecycle, the industry sector, and the level of testing (component, integration, system or acceptance testing). Some common objectives are:

- find defects (in component, integration and system testing);
- gain confidence about the level of quality of a system, and reporting to others, i.e. providing information about the software or system being tested;
- providing information for decision-making (typically about the software quality, what works, what fails, etc.);
- preventing future defects, by learning from root cause analysis and improving processes (defects found in test design, test execution or review).

Dynamic and Static Testing

Testing can be described as dynamic or static. Dynamic testing is what we normally think of as testing, i.e. running test cases through the system. It is described as dynamic because the software is being executed (dynamic execution). Static testing is where the software or its artefacts are examined, without any test cases being run. Both static and dynamic testing have similar objectives: to find defects, measure aspects of the system and hence оттука to gain confidence in the system, and to find ways to improve processes (testing and development processes) and thereby на тој начин to prevent similar defects from occurring се случуваат in the future. The objective of testing may depend on the level of testing. In development the main objective may be to find as many defects as possible. In acceptance testing, the objective may be to gain confidence.

Static testing can be done through reviews of documents or through the use of special static analysis tools, which examine the code we are interested in. Dynamic testing can be carried out manually (with the tester typing the inputs into the system) or automatically through the use of a test execution tool. Dynamic testing uses test cases and runs the software; static testing does not. Static testing can be done early in the lifecycle, but so can the design of dynamic test cases. When test inputs are derived from a specification, it is referred to as “black box” testing; when test inputs are derived from the code itself or a model of the code; it is referred to as “white box” testing (also referred to as “glass box”).

Note that the thought мисловните processes and activities involved in designing dynamic tests can lead to the identification of defects in documents (the test basis) before the tests are executed. Since these activities can be undertaken before the program code is written this can help to prevent some defects occurring in the code.

Debugging and Testing

Debugging is different to testing, though many people seem to lump them together. Testing can find defects (e.g. through a review, a form of static testing) or can generate failures (by inputting one or more values that cause the system to produce an incorrect result in a dynamic test). Debugging is the process of fixing the defect that was found or that caused the failure. Debugging is a development activity, not a testing activity.

1.3 Seven Testing Principles

Learning Objective:

LO-1.3.1 K2 Explain the seven principles in testing.

Terms

Exhaustive testing.

A number of testing principles have been suggested over the past 40 years and these offer general guidelines common for all testing. They are fundamental truths that apply in more or less any testing context.

Principle 1 -Testing shows presence of defects

If we run a test and it finds a defect, this is useful – that defect could be fixed, and then our software would be (slightly *малку*) improved.

If we run a test and don't find any defects, we cannot conclude that there aren't any – it is much more likely that our test wasn't good enough to find the ones that are there.

There are many reasons for testing including building confidence in the software under test, demonstrating conformance to requirements or a functional specification, to find defects, reduce costs, demonstrate a system meets user needs and assessing the quality of the software. However, one reason that is not valid (but often used erroneously *погрешно*) is to *prove* that the software is correct. This is wrong simply because it is not possible to prove that a software system is correct. It is not possible to prove a system has no defects. It is only possible to prove that a system has defects - by finding some of them!

Principle 2 - Exhaustive testing is impossible

Exhaustive testing is defined as exercising every possible combination of inputs and preconditions. This usually amounts to an impractical number of test cases. For most applications it is relatively easy to identify millions of possible test cases (not necessarily all desirable test cases). For example, an application that accepts an input value corresponding to an employee's annual salary may be expected to handle figures in the range £0.00 to £999,999.00. Given that every different input value could be made a separate test case this means that we can identify 100,000,000 test cases (the number of 1p's in the range) just from this one input value. Of course, no one would reasonably expect to have to try out every one of these input values but that is what exhaustive testing would require. Exhaustive testing would also require each of these 100,000,000 input values to be tried in different combinations with other inputs and with different preconditions (such as data values in a database). For even simple applications the number of test cases required for exhaustive testing is likely to run into millions of millions of millions.

It may be possible to test every combination of precondition and input for a very small system, but it is completely infeasible and impractical to do it for any realistic system. Risk and priorities are used to decide what to test and what not to test.

Principle 3 - Early testing

To find defects early, testing activities shall be started as early as possible in the software or system development lifecycle, and shall be focused on defined objectives.

Note the terminology used here. In the syllabus ISTQB state that "testing activities shall be started as early as possible". This sounds more like a command than a principle in our view! However, the principle remains: it is generally better to start testing activities as early as possible in a project so the defects that are found can be fixed more cheaply than is usually possible than is the case when they are fixed later in the development lifecycle.

Testing should also focus on the defined objectives for this testing, which may be different depending on the level of testing. (Test objectives were discussed in section 1.2 under 'Objectives of Testing'.)

Principle 4 - Defect clustering

Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contain most of the defects discovered during pre-release testing or are responsible for most of the operational failures.

Again, note the terminology used "testing effort shall be focused proportionally". We should keep in mind that this is a principle that probably applies in most cases but not all (see the discussion on the testing principle "Testing is context dependent" below).

"Bugs are social creatures – they hang around together". It seems counter-intuitive but defects and failures do tend to cluster together, and this has been shown by a number of studies. For example, an IBM study found that 4% of the modules of an operating system contained 38% of the defects. In another study, 7% of the modules contained 57% of the defects. [p280, Applied Software Measurement, Capers Jones, McGraw-Hill, 1991.]

It makes sense then that testing should be more thorough where it is most needed. We may be able to predict that some modules (of parts of a system) will contain more defects. This may be because they are more complex or because they do more, or were particularly difficult to implement for some reason. Once we start test execution we may discover that some parts of the software do contain more defects than others - if this is different to our predictions then we should consider changing the focus of the testing concentrate on the software with most defects.

Principle 5 - Pesticide paradox

A pesticide is a chemical that is used to kill pests *шtetnici* (i.e. insects) that would otherwise destroy crops. However, if a pesticide is over-used, the insects evolve to be resistant to the pesticide, so the chemical loses its effectiveness.

The first time a particular test is run is when it is most likely to generate a failure and reveal a defect. Once that defect has been fixed, this test becomes a re-test and is likely to pass. So the ability of an individual test to find a defect is lower after a test has been run once. So, a set of tests that is run many times, with any defects fixed, is less and less likely to find new defects. Hence *Оттуда* the software becomes "immune" to these tests and the developers become wise about where you are focusing testing effort.

Principle 6 - Testing is context dependent

Software from different industries (e.g. safety-critical versus entertainment) does not and should not be tested to the same level of thoroughness or formality. The best testing is the most appropriate testing – for the company and the situation. One organisation may test their software much more thoroughly than another (or conversely, much less thoroughly) but this does not mean one organisation is doing good testing and the other bad testing. The thoroughness of the testing undertaken depends on many aspects of the context.

Principle 7 - Absence-of-errors fallacy

It is easy to assume that having no defects must be the best thing for a system. But this is not the case. A system that has a few minor defects but is otherwise very helpful and useful to its users is better than a supposedly defect-free system that doesn't provide a useful service, or which is so hard to use that its functionality cannot be used easily.

Just think of a web site that is very hard to find your way around, or one that keeps asking you to re-enter information you have already entered. It may not have any defects (that you have encountered) but that doesn't make it a good web site. A web site that is very easy to use but occasionally overlaps a box with another does have an easily visible defect, but it is a better web site even with its defect.

1.4 The Fundamental Test Process

Learning objective:

LO-1.4.1 K1 Recall the five fundamental test activities and respective tasks from planning to closure.

Terms

*Confirmation testing, re-testing, exit criteria, incident, regression testing, test basis, test condition, test coverage, test data, test execution, test log, test plan, test procedure, test policy, test suite *накѐм*, test summary report, testware.*

The fundamental test process is a model of testing activities performed within a project that describes five distinct sets of activities. For a given test effort they may be followed more or less sequentially but there is likely to be some overlap *преклопуваат*. The extent to which it is applied in different organisations and across different projects within the same organisation will vary *се разликуваат-варираат* considerably (testing is context dependent after all!). Where testing is required to be undertaken *преземен* very formally (such as in safety critical systems) all the activities may be carried out and documented in detail. Where *Каде* testing is performed less thoroughly many of the activities may be undertaken concurrently or missed out altogether with little or no documentation.

The purpose of the fundamental test process is not to prescribe how testing should be done but rather to give us a reference model that can be adapted to different situations.

1.4.1 Test Planning and control

Test planning and control are related to the management of testing on the project. The test plan should ideally be influenced by the overall test strategy (also called the test approach) and possibly a company-wide test policy.

There are two approaches to testing that we need to consider at this stage:

- **Preventative:** to start activities as soon as possible to prevent problems arising;
- **Reactive:** responding to the situation once the software is delivered.

A preventative approach to testing typically involves much planning and preparation work (for example static testing and designing dynamic tests) so we are ready to start test execution as soon as the software is becomes available. A reactive approach typically involves little or no planning or preparation work, rather we wait until the software is available before undertaking any amount of testing work. Reactive approaches are not generally recommended in favour of preventative approaches but we can find ourselves in situations where a reactive approach is our best option.

The test planning activity produces a test plan specific to a level of testing (e.g. system testing). These test level specific test plans should state how the test strategy and project test plan apply *се применуваат* to that level of testing and state any exceptions to them. When producing a test plan, clearly define the scope of the testing and state all the assumptions *претпоставки* being made. Identify any other software required before testing can commence (e.g. stubs & drivers, word processor, spreadsheet package or other 3rd party software) and state the completion criteria to be used to determine when this level of testing is complete.

Note that the exit criteria from one stage of testing may form the entry criteria to the next stage, but there may be others not directly related. For example an entry criterion to System Test may be that the test environment has been set up, but this is an environment only for system testing, not used in earlier stages of testing.

Test control is where we keep track of what is actually happening, to compare it to the plan that we made earlier. Test control includes monitoring of various factors, relating to the testing as well as what we are testing. We will keep track of the number of test conditions, test cases, scripts, etc. We will count the defects that have been found. We will keep an eye on our

schedule, and we may need to do something different to our plan when things don't work out as we thought they would.

Test control has the following major tasks:

- measuring and analysing results;
- monitoring and documenting progress, test coverage and exit criteria;
- making decisions and initiating corrective actions.

1.4.2 Test analysis and design

This part of the fundamental test process is where test objectives are translated into test conditions and tests are designed to cover the most important test conditions. In other words, we develop a set of tests that are geared to achieving our test objectives (e.g. many and detailed tests to find as many defects as possible or a few high level tests to establish basic confidence in the software).

The first step in test analysis is to review the documents that describe what the system should do (e.g. requirement specifications, system specifications, architectural design specifications, detailed design specification, use cases, and the source code). A more general term for these documents is 'test basis', i.e. what the tests are based on. Different levels of testing will typically use different test basis documents. Note that reviewing a document can be done very early in the lifecycle – as soon as the test basis document becomes available. The purpose of this review is to determine the testability of the test basis (i.e. make sure it is possible to test what is described) and consequently the test objects (the software that is to be tested).

When reviewing test basis documents, test analysts should also review the software integrity level and risk analysis reports where these are available. The software integrity level is the degree to which the software is required to comply with certain software characteristics (such as complexity, risk, safety, security, performance, reliability and cost). The choice of characteristics and measurement criteria should be made by the stakeholders to reflect the importance of the software to them.

The next step is to identify test conditions, that is, 'things' that could be tested. These should be prioritised to ensure that the most important ones are tested first and in preference to less important test conditions.

Formal testing techniques mostly concentrate on identifying test conditions. Sometimes a brainstorming session is also good for this. When we initially brainstorm we will think of a few conditions - that in turn will trigger more, but remember that the first 50% produced is unlikely to be the best 50%. Therefore, if you need 100 test conditions, identify 200 (or even 1000) and pick the best 100 test conditions.

Having identified the most important test conditions the next step is to design test cases. This requires specification of the exact and specific inputs that will be entered to the system and the exact results that will be checked. The test cases should then be prioritised. Test cases that exercise the most important test conditions will be effective (meaning that they will have the potential to find defects that are important to find and fix).

Designing good test cases is a skill. To be exemplary a test case should exercise several test conditions but to be economic and evolvable it should not be too big or too complex.

Predicting the expected outcome is part of test design. The term outcome is used in preference to output because the outcome comprises everything that has been output and what has been changed, deleted, and not changed.

An expected result is what the software is supposed to produce when a test is executed. We usually refer to the expected outcome (or results) rather than the expected output since outcome is a more encompassing word. Outcome includes everything that has been created, changed, or deleted and also includes things that should not have changed. In other words it is the difference between the state of the system and its environment before and after executing the test.

It is important that expected results be specified in advance of tests being executed though not doing so is a fairly common practice. If we do not specify the expected results for a test the intention would be to verify the actual results by viewing them at the time we execute the test. This has the advantage of reducing the amount of work we have to do when specifying the tests (less documentation, less effort). However, this approach has the disadvantage of being less reliable. There may be a subconscious desire to see the tests pass (less work to do - no fault report to write up). It is also a less rigorous approach since a result that looks plausible may be thought correct even when it isn't wholly correct. Calculating an expected result in advance of test execution and then comparing the actual result with this is a much more reliable approach. It also means that the tester does not have to have the knowledge that would allow him or her to properly verify test outcomes.

Expected results should be derived from a specification of what the system should do (for example a requirement or functional specification). In other words the expected results are determined by considering the required behaviour of the system.

Traceability следливост between test conditions and test cases should be maintained. This will make it easier to identify appropriate subsets of tests to execute and also to identify which test conditions are associated with failed test cases.

Some approaches to testing lead testers to consider the structure of the system when designing test cases, in particular by examining the source code. While this is a good approach to identify test inputs it is vital that the expected result is not derived in this way. The expected result should be derived from the required behaviour of the system, not its actual behaviour. Basing expected results on constructs and values within the source code will have the effect of generating test cases that test that the software does what the software does, whereas what we really need is test cases that test that the software does what the software should do.

For most systems it is possible to predict the expected results for any test case but there are a few systems where this is not the case, for example, systems that predict situations or perform long and complex calculations that cannot practically be performed manually. The Oracle Assumption is the name given to the assumption that it is possible to predict the expected results.

It is frequently necessary to design not only individual test cases but also whole sets of test cases each with different objectives. For example, regression tests, performance tests, and detailed tests of a particular function.

During the test analysis and design activity the test environment is also designed, and infrastructure issues are addressed, such as acquiring tool support for testing.

1.4.3 Test implementation and execution

Test Implementation involves making the test cases a reality (finalising the test cases, e.g. finishing any outstanding specification work), and includes writing test procedures or test scripts, creating or acquiring the test data and implementing the expected results. These are pre-requisites prior to test execution.

The test cases are organised into groups and a sequence of execution is decided – the document that describes the order of running of the tests is the test procedure, also commonly called a test script. Test scripts may be designed for either manual or automated execution. Tests may be grouped together into test suites for specific objectives (e.g. regression test, performance test, depth test of a feature).

The environment needs to be set up and verified at this stage, and any additional software needed to run the tests, such as test harness *искористување* or scaffold *скеле* needs to be ready.

The purpose of test execution is to execute all of the test cases (though not necessarily all in one go). This can be done either manually or with the use of a test execution automation tool (providing the test cases have been designed and built as automated test cases in the previous stage).

The order in which the test cases are executed is significant. The most important test cases should be executed first. In general, the most important test cases are the ones that are most likely to find the most serious defects but may also be those that concentrate on the most important parts of the system.

There are a few situations in which we may not wish to execute all of the test cases. When testing just defect fixes we may select a subset of test cases that focus on the fix and any likely impacted areas (most likely all the test cases will have been run in a previous test effort). If too many defects are found by the first few tests we may decide that it is not worth executing the rest of them (at least until the defects found so far have been fixed). In practice time pressures may mean that there is time to execute only a subset of the specified test cases. In this case it is particularly important to have prioritised the test cases to ensure that at least the most important ones are executed.

If any other ideas for test conditions or test cases occur to you at this stage they should be documented so that they can be considered for inclusion either immediately or later.

Part of test execution is to record the versions of the software under test and the test specification being used. Then for each test case we should record the actual outcome and the test coverage levels achieved. *постигне* for those measures specified as test completion criteria in the test plan. In this way we will be marking off our progress. This is referred to as a test record or the “test log”. Note that this “test record” has nothing to do with the recording or capturing of test inputs that some test tools perform!

The actual outcome should be compared against the expected outcome and any discrepancy found logged and analysed in order to establish where the defect lies. It may be that the test case was not executed correctly in which case it should be repeated. The defect may lie in the environment set-up or be the result of using the wrong version of software under test. The defect may also lie in the specification of the test case: for example, the expected outcome could be wrong. Of course the defect may also be in the software under test! In these cases the defect should be reported and the test case will need to be executed again after the defect has been fixed.

The test record should be detailed enough to provide an unambiguous *недвосмысленна* account of the testing carried out. It may be used to establish that the testing was carried out according to the plan.

1.4.4 Evaluating exit criteria and reporting

The exit criteria were set in the Test Plan, and are now checked to see whether or not we have achieved what we planned in our testing. Each test level checks its own exit criteria, which would be different for different test levels or objectives. For example, the exit criteria for component testing may include a code coverage measure, and the exit criteria for acceptance testing may include an objective that there are no high priority defects outstanding, or that any that have not been fixed have defined workarounds.

The test log is the record of what actually happened in testing, so this is checked against the exit criteria.

If the exit criteria have been met *исполнети*, then we have (by definition) finished this stage of testing.

If we have not met all the exit criteria, then a decision needs to be made. This could be, for example:

- we need to write more tests to increase coverage to the percentage specified for this component
- we will accept the current coverage, i.e. modify our exit criteria (this should only be done with good justification, and should be documented)

As part of this activity, we will also write a Test Summary Report to summarise what has happened with the testing. If we are using an iterative development model, there may be a Test Summary Report produced for each iteration, as well as a fuller Test Report at the end of the project.

The Test Summary Report will go to the stakeholders, and may be used in making a decision whether or not to release the system.

The contents of a Test Summary Report are given in IEEE 829, and this is included in Section 5.

1.4.5 Test closure activities

Once a project has finished, and a system has been released, that is not the end of everything from a testing point of view. It is important that the “loose ends” are tied up neatly. Test closure activities may also be performed at project milestones such as when a system is released, when a project is completed or cancelled, at an intermediate milestone, or after a maintenance release has been completed.

The activities performed include:

- Checks are made that the correct deliverables *испораки* have been produced and delivered to the right destinations. For example if formal acceptance *прифаќање* is required, i.e. a sign-off of approval, ensuring that the signatures are correctly filed for future reference if needed.
- Incident reports should also be checked. Any that are still outstanding to be fixed, i.e. have been deferred, should be forwarded to the next release or project to be fixed, or should be closed if they will not be fixed.
- The testware should be finalised, archived and handed over to those who will be responsible for maintaining it. This is particularly critical for automated testware which will be used for regression testing of the system when future changes are made. It is worth spending time ensuring that the tests to be preserved are a good set, i.e. will do the regression testing effectively and efficiently.
- The test environment and any test infrastructure should either be closed down or handed over for reuse by another part of the organisation.
- A post-project review should be held that specifically looks at testing issues (among other things), to recognise what was done well (and should be done in a similar way next time) and those things that encountered problems, so that they can be done differently next time. Lessons can be learned at this point that will otherwise soon be forgotten. It is also important that the new ways of doing things are communicated to those who will do the testing next time. The test maturity *зрелост* of the organisation is raised if this process is done well.

1.5 The Psychology of Testing

Learning Objectives:

- | | | |
|----------|----|-------------------------------------------------------------------------|
| LO-1.5.1 | K1 | Recall the psychological factors that influence the success of testing. |
| LO-1.5.2 | K2 | Contrast the mindset of a tester and of a developer. |
-

Terms

Error guessing, independence of testing.

Different Mindset

The point of view (or mindset) of a developer is different to the mindset of a tester. When a developer is writing new code (or modifying code), their focus is on understanding what the code should do and writing something that will do what is required. There may be technical constraints that they need to work within that they also need to consider, but their main aim is to get it to work.

The tester's mindset is not quite the opposite, as testers are also concerned with understanding the requirements. But the tester's approach is to try and break it, to consider

what could go wrong, i.e. to look at the code or system from what could be described as a negative point of view.

Should developers test their own work? Is a developer capable of testing their own work? The answer is Yes to both of these questions! However it can be difficult to “change gear” and think “testing” when you are firmly in “developer” mode. Some developers are very good at testing their own work; others are not so good. Hence Отпуска having someone independent of the developer to test can be more effective at finding defects. It has been estimated that most people only find half or less of their own defects. This is because we are “blind” to some extent to our own mistakes – we see what we intended to write instead of what we actually wrote.

Levels of independence

This is re-visited in more detail in section 5 which is looking at the organisation of testing and test teams. Testing done by someone who has not been involved in the development of the software under test is likely to be better (i.e. find more defects) than testing done by someone who has been involved in the software's development. This is because the person who has been involved in the development process will have a restricted view of the software, a developer's view. In this case any assumptions предположения made during the development process are likely to be carried over into testing. The other person will be able to view the software independently of the development process and be able to challenge any assumptions the developers made.

There are many levels of independence, that is, ways of achieving a greater or lesser amount of independence in software testing. It is important to appreciate that the independence is most required at the specification stage of the testing process. It is here that the test cases are designed and it is the design of the design test cases (more specifically, the identification and selection of the test conditions) that governs владеет the quality of the test cases. People who say that programmers should not test their own code often miss this point. What they should say is that programmers are not the best people to specify all of the tests for their own code. However, programmers should contribute to the specification of the test cases since they can contribute a good technical understanding of the software.

There are good reasons why programmers should test their own code. Perhaps the best of these is that it is cheaper. A programmer who finds a defect in his or her own code can quickly fix it and execute the test again to confirm the fix. The fact that a defect was found and fixed does not need to be documented. If someone else were to run the test, they probably would need to document it as a safe and secure way of communicating the details to the programmer. The programmer would then need to reproduce the failure and fix the defect. The defect report would then be updated and make its way back to the tester who would repeat the test to confirm the fix was correct. This takes much more effort in total and yet no more has been achieved: one defect has been found and fixed.

We will achieve no independence if only the person who wrote the software specifies the tests. If another developer from the same team were to specify the tests, a little independence is achieved достигнуто. More independence can be achieved if someone outside of the development team specifies the tests (such as a test team). Further independence can be achieved by having an outside agency undertake the testing though this in itself introduces some different problems. Perhaps the greatest level of independence can be achieved by having a tool generate test cases but these are not likely to be particularly good quality tests.

Several несколько levels of independence can be defined. The following levels of independence are identified in the Syllabus and are shown here in order from low independence to high independence:

- **person who wrote the software** designs the tests (low or no independence)
- **another person** from the development team designs the tests (could be buddy system or pair programming, or it could be a tester on the development team responsible for testing all developer code)
- **a different organisational group** (e.g. an independent test team within the organisation, such as a System Test Team, Performance Test Team or User Acceptance Test Team)

- **a different organisation or company**, e.g. outsourced to a third party testing consultancy, or tests design and run by an off-shore organisation.

Independence is not a panacea лек – it is not a replacement for familiarity, for example. Someone who is deeply familiar with a component may be able to find defects that someone with a more superficial knowledge might miss.

Perception of testing

The objectives for testing should be stated. Sometimes testing will be directed at one objective, such as trying to find defects. At another time the main objective of testing may be to gain confidence that it will support a business process. Depending on what the objective is, the testing will need to focus on whatever that objective is. Testing that is very time-constrained may be conducted in a different way to a test effort that has the time to be more thorough. People tend to meet the objectives that apply, or at least that they think are applicable применливо.

Testing can be seen as a destructive process. Looking for defects in something has a negative connotation; it implies that the something is faulty to start with. The point is, when it comes to software, it usually is faulty in some way! (Of course developers are inclined to believe the best of their software, because they created it, it is their baby.)

Care should be taken when communicating defect information to developers and managers. Dashing up to a developer and saying "You fool, you did this wrong" is not likely to encourage him or her to investigate the problem. More likely the developer will go on the defensive, perhaps arguing that it is not a defect but it is the tester who does not understand it. A more successful approach may be to approach the developer saying "I don't understand this, would you mind explaining it to me please?" In demonstrating it to the tester the developer may then spot the defect and offer to fix it there and then.

Cem Kaner (co-author of "Testing Computer Software") says that the best tester is not the one who finds the most defects but the one who manages to have the most defects fixed. This requires a good relationship with developers.

Developers are not the enemy!

One of the most encouraging trends in recent years is the closer collaboration and cooperation of testers and developers. Extreme Programming and many other Agile methods put testing very much up-front in the development process, so developers are becoming more interested in testing, and (hopefully) more appreciative благодарни of what testers do.

Sometimes, especially when testers are an independent group, a level of animosity can build up between testers and developers, but this is not always the best thing, and if taken to an extreme and be very damaging to the whole organisation.

Ways to improve communication (and also relationships) include:

- remember that you have common goals, i.e. better quality systems in the end, and start with cooperation rather than confrontation;
- communication about defects should be factual, objective and neutral (expressed in the 3rd person rather than the 2nd person – "this input produces an output that does not correspond to .." rather than "you didn't process this input correctly");
- try to understand how the person you are communicating to might feel about what you are saying, and be sensitive to their feelings;
- use "active listening" to confirm that your message has been correctly understood, and that you have correctly understood what the other person has said to you (e.g. echo back what they said, and ask them to do the same.)

1.6 Code of Ethics

Introduction

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use. Recognising the ACM and IEEE code of ethics for engineers, the ISTQB® states the code of ethics listed below. These seek to identify right from wrong and good from bad in a number of areas that may concern testers.

Public

Certified software testers shall act consistently with the public interest.

Client and employer

Certified software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

Product

Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible.

Judgement

Certified software testers shall maintain одржување integrity and independence in their professional judgment.

Management

Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing.

Profession

Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest.

Colleagues

Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers

Self

Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Session 2

Testing Throughout the Software Lifecycle

Contents

- 2.1 Software Development Models
- 2.2 Test Levels
- 2.3 Test Types (Targets of Testing)
- 2.4 Maintenance Testing

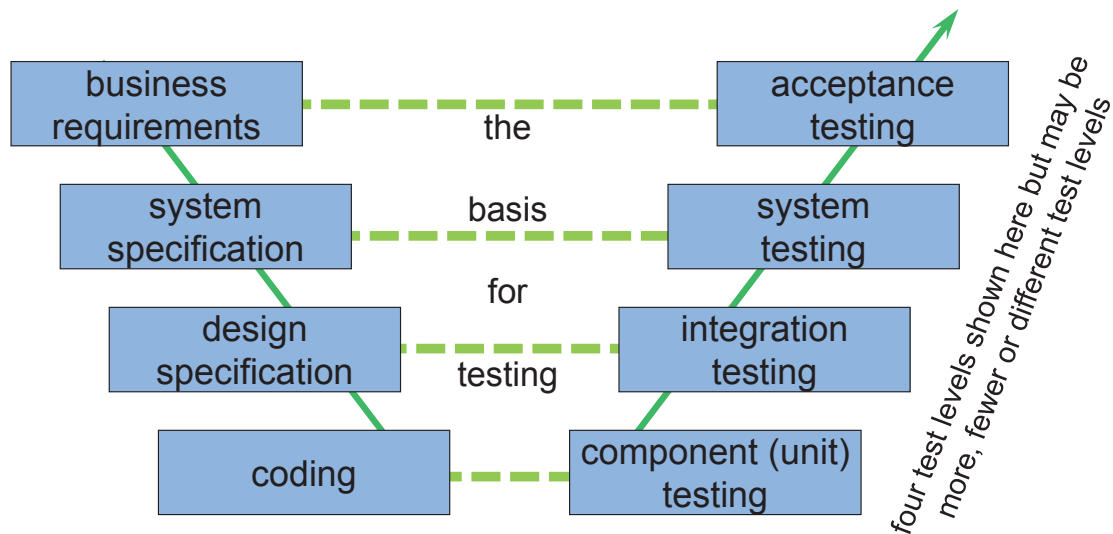
2

Software Development Models

- two types – sequential and iterative-incremental
- sequential (e.g. waterfall, V-model)
 - product is designed and built in one delivery
 - system analysis and design all at the beginning, test execution at the end
 - often produces detailed documentation
- iterative–incremental (e.g. prototyping, RAD, RUP, agile)
 - product is designed and built - many deliveries
 - system analysis, design, code and test are integrated
 - documentation is produced only if necessary and often is lightweight

3

V-Model: Common Type



4

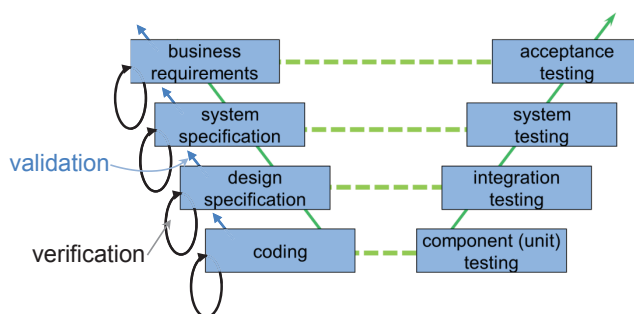
Verification and Validation

Similarities

- both forms of testing
- typically performed by reviews of work products*
 - by individuals or groups

Differences

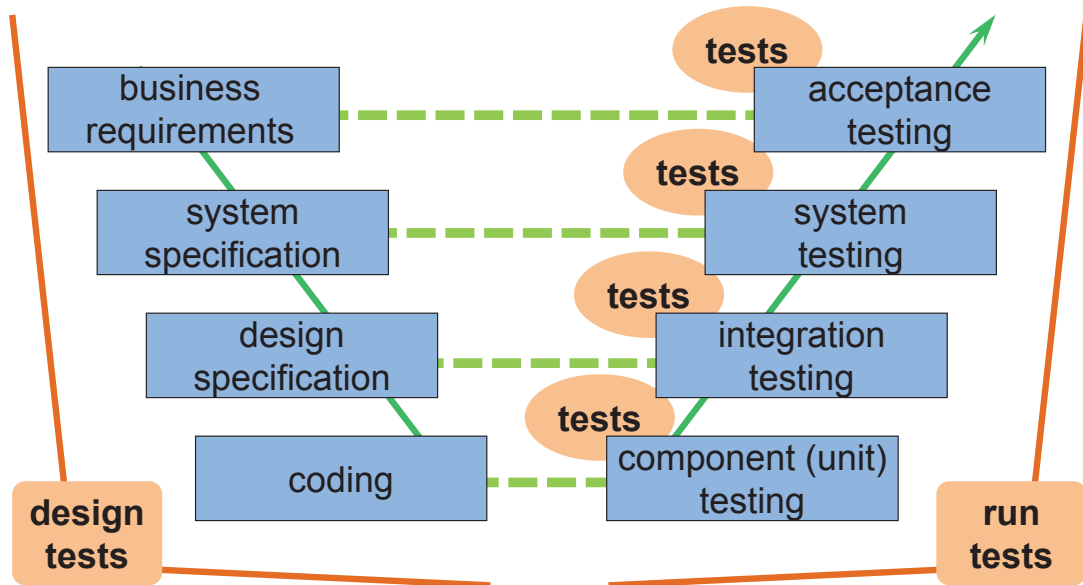
- focus:
 - verification focuses on the process of producing the work product
 - i.e. is it a good specification / code?
 - validation focuses on the use from the users' perspective
 - i.e. is this what the user wants?



* CMMI and IEEE/IEC 12207
'Software Lifecycle Processes'

5

V-Model: Early Test Design



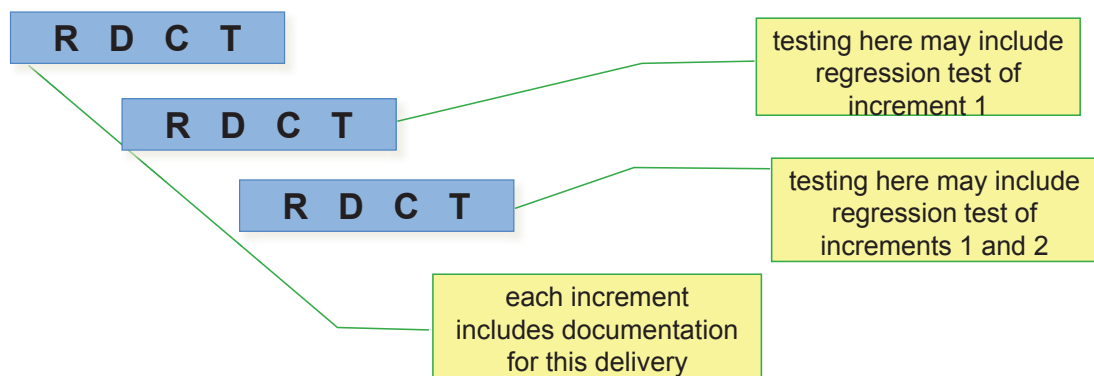
6

Iterative-incremental Development Models

- product developed in short cycles
 - each cycle includes requirements, design, code & test
 - various test levels can be applied to each cycle
 - regression testing becomes increasingly important

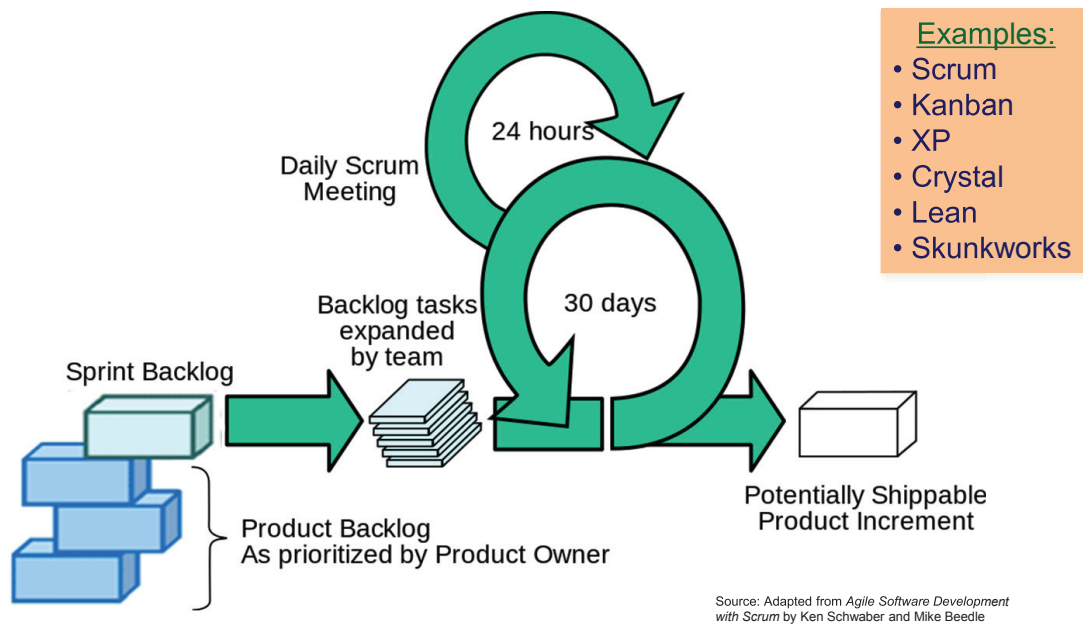
Examples:

- RAD
- RUP
- DSDM
- Prototypes
- Agile



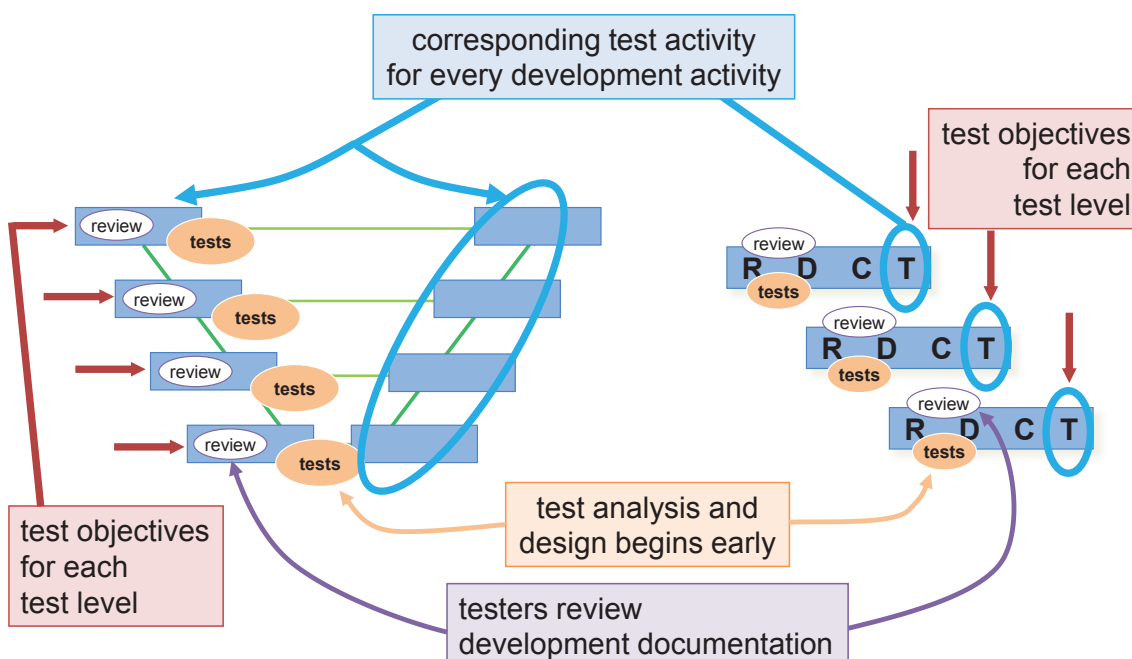
7

Agile Development (Iterative-incremental)



8

Good Testing Within a Lifecycle Model



9

Contents

- 2.1 Software Development Models
- 2.2 Test Levels
- 2.3 Test Types (Targets of Testing)
- 2.4 Maintenance Testing

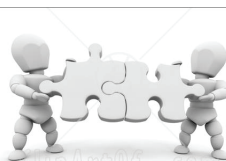
10

Test Levels: What Are They?

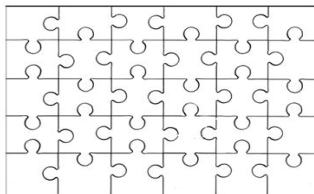
- component testing



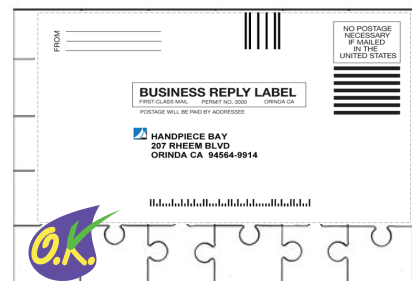
- integration testing



- system testing



- acceptance testing



11

Component Testing (Unit, Module, Program Testing)

- lowest level testable item
 - tested in isolation if possible
 - may use stubs *никулци* and/or drivers
- usually performed by programmer
 - within a development environment
 - support with tools such as unit test framework tool or debugging tool
- defects fixed as soon as they are found
 - normally no formal recording of defects



test basis

- program spec
- design spec
- code

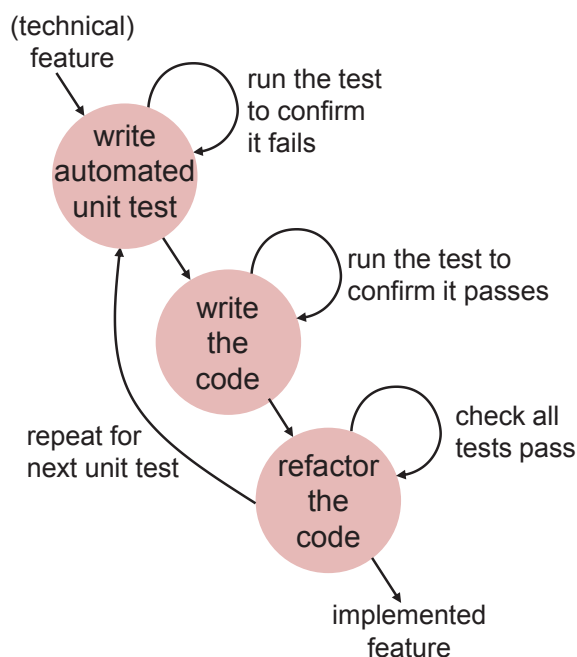
test objects

- components
- programs
- database modules

12

Test-Driven Development (TDD)

- Agile development approach to component testing
- write an automated unit test
 - confirm it fails
- write the code
 - one test at a time
 - confirm test passes
- refactor the code
 - to ensure quality
- all tests run again
 - to prevent regression
- repeat for next unit test



13

Test Harnesses / Unit Test Framework (Developer)

- used to exercise software that does not have an interface (yet)
- facilitates *олеснува* testing by simulating an environment in which the tests will be run
 - may use mock objects such as stubs and drivers
 - may also use simulators
 - ▶ particularly where testing in real environment would be too costly or dangerous
- aids component testing in parallel with building the code
- may be custom-built
- includes debugging support



14

Dynamic Analysis Tools (Developer)

- finds defects that are only evident when the software is / tests are run
 - provide run-time information on software
 - ▶ allocation *распределба*, use and de-allocation of resources, e.g. monitoring memory use helps find 'memory leaks'
 - ▶ highlight unassigned pointers or pointer arithmetic defects
- useful when testing middleware
- typically used in component testing and component integration testing



15

Modelling Tools (Developer)

- validate models of the software
 - finding defects in data, state and/or object models
- aids test case generation
 - where test cases are based on the model
- benefits
 - defects are found at the earliest opportunity
 - ▶ saves time, money and accelerates process



16

Integration Testing

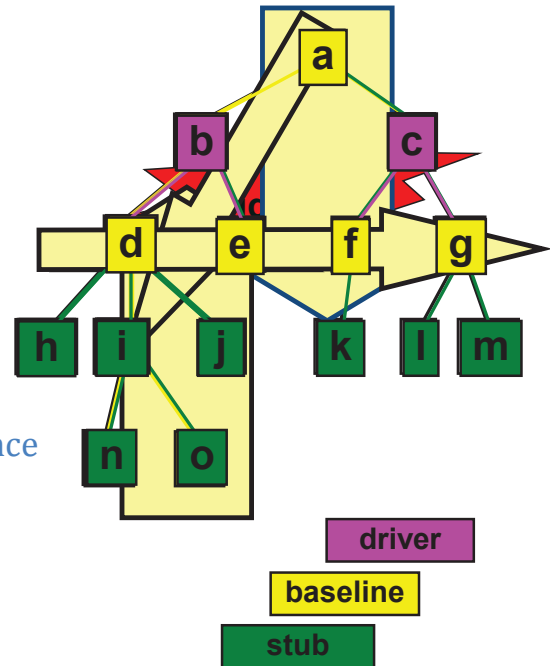
- multiple levels of integration
- component integration testing
 - interfaces/interactions between components
 - performed after/during component testing
- system integration testing
 - interfaces/interactions between systems & hardware/software
 - usually performed after system testing
 - usually performed by independent test teams
 - problems can exist when multiple organisations involved or cross-platforms are used



17

Integration Strategies

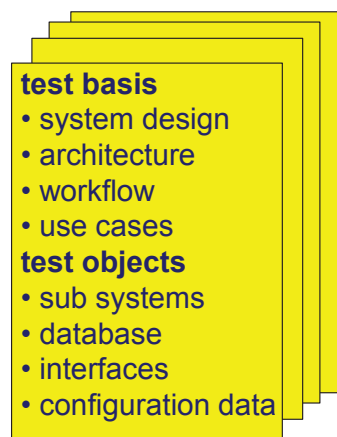
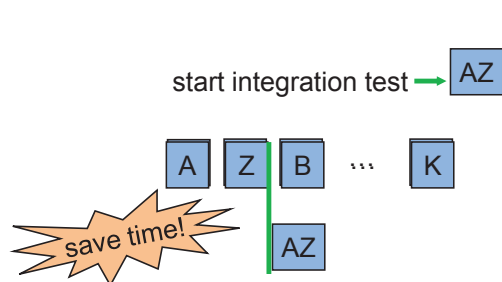
- how much at once?
 - big-bang (not recommended)
 - incremental (reduces risk of late defect discovery)
- incremental based on
 - system architecture
 - ▶ (e.g. top-down, bottom-up)
 - functional tasks
 - transaction processing sequence
 - other aspect



18

Considerations for Integration Testing

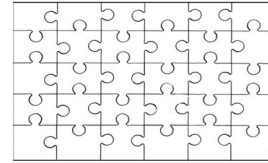
- testers should be involved early
 - understand the architecture
 - plan the integration test order first
 - ▶ this can determine the build order
 - ▶ most efficient testing



19

System Testing

- testing the system as a whole
 - functional, non-functional and structure tests are considered
 - often performed by an independent test team
 - problems may occur with interfaces if previous stage of testing not performed well
- system test environment is essential
 - realistic and representative to correspond to production environment
 - minimises environmental defects not found



20

Considerations for System Testing

test basis

- specifications (req., system & functional)
- use cases
- risk register
- manuals
- experience

test objects

- manuals
- configuration data
- system

testers may need to work with incomplete or undocumented requirements



- designing system level tests
 - specification-based* testing first to ensure functionality works
 - structure-based* testing second to assess thoroughness

* covered in Session 4 Test Design Techniques

21

Test Execution Tools

- interface to the software being tested
 - run tests as though run by a human tester, simulates user interaction, keystrokes, mouse movements
- test scripts in a programmable language
- data, test inputs and expected results held in test repositories
- most often used to automate regression testing
 - can also be useful to record tests during an exploratory test session

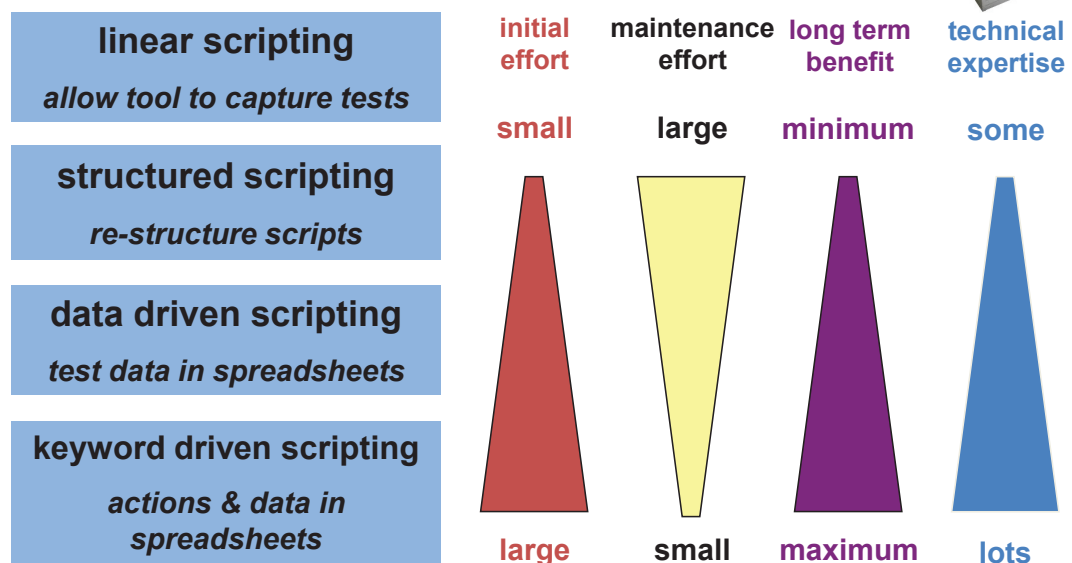


special considerations

significant effort in scripting is required to achieve the significant benefits from this type of tool

22

Considerations for Execution Tools



23

Test Comparators

- detect differences between actual test results and expected results
 - screens, characters, bitmaps
 - masking and filtering
- test running tools normally include dynamic comparison capability
- stand-alone comparison tools for files or databases (post execution comparison)
- a test oracle may be used as a basis for comparison



24

Acceptance Testing

- clear objective – to gain confidence
 - in any aspect of the system
 - should not find (m)any defects
 - is it ready to deploy?
- responsibility of customers, users, stakeholders
 - may use different test environment
 - can perform any test they want



25

Considerations for Acceptance Testing

test basis

- requirements (user & system)
- use cases
- business processes

- risk register
- manuals
- experience

test objects

- forms & reports
- user procedures
- processes
- complete system

- this might not be the final stage of testing, e.g.
 - large-scale system integration acceptance testing of one system
- acceptance testing may occur at various stages, e.g.
 - testing COTS when installed
 - testing usability of a component
 - testing new functionality prior to release to system test

26

Types of Acceptance Testing

user acceptance testing

- fitness for use by business users

operational (acceptance) testing

- backup/restore, disaster recovery, user management, maintenance tasks, security checks, (vulnerabilities), data load

contract and regulation acceptance testing

- regulations, e.g. government, legal, safety
- contract's acceptance criteria

alpha and beta (field) testing

- feedback from potential / existing customers for COTS package developers, before commercial sale
- alpha: on developer site
- beta: off developer site













other acceptance testing terms

- factory or site acceptance test
- before / after being moved to customer site

27

Tailoring of Test Levels

- project or system may determine test levels

	small internal system	commercial off the shelf COTS	software sold to external customers	safety critical internal system
Component testing				
Integration testing				
System testing				
User acceptance				
Alpha/Beta acceptance				

Note: the test levels will depend on the context

28

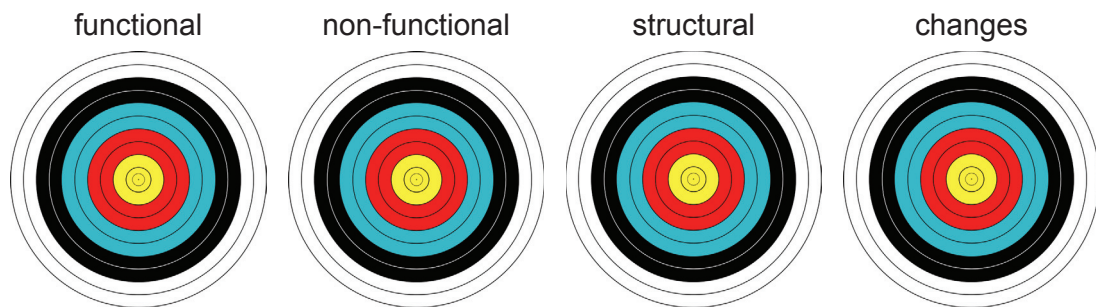
Contents

- 2.1 Software Development Models
- 2.2 Test Levels
- 2.3 Test Types (Targets of Testing)**
- 2.4 Maintenance Testing

29

Test Types – What Are They

- 4 types/targets of testing – aspect focused on (objective)

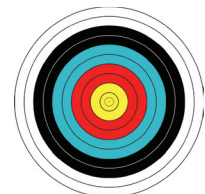


- models or text can be used for these targets
 - functional: process flow, state transitions, req. or design spec.
 - non-functional: performance or usability model, req. spec.
 - structural: control flow, menu structure, program spec. or code

30

Testing of Function / Functional Testing

- function: “what” a system does
 - external behaviour of the software (black-box)
 - tests are produced from specifications or knowledge
- performed at all levels
- examples include:
 - report is produced
 - discount applied to members
 - system prevents unauthorised access



31

Security Tools / Security Testing Tools

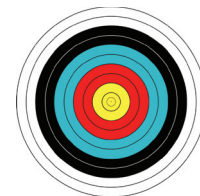
- security tools
 - e.g. virus checking, firewalls
 - not strictly testing tools but can assist in security testing
- tools that support security testing
 - searching for vulnerabilities
 - ▶ e.g. denial of service attacks, probe for open ports, password files
 - can attack networks, support software, application code, database



32

Non-functional Testing

- non-functional: “how well” the system works
 - may be performed at all levels
 - measured on a variable scale (e.g. response time)
 - tests are produced from specifications or knowledge
- external behaviour (black box)
- non-functional quality characteristics*:



also known as quality attributes

- usability
- efficiency
- reliability
- portability
- maintainability

sub-characteristics:
<ul style="list-style-type: none">• <u>performance</u>• resource use
<ul style="list-style-type: none">• maturity• robustness• recoverability

test types:
<ul style="list-style-type: none">• <u>load testing</u>• <u>stress testing</u>

* as defined by ISO 9126: Software Product Quality (superseded by ISO 25000)

33

Performance Testing Tools

- performance, load and stress tools
 - drive application via user interface or test harness
 - simulates realistic load on the system, application, a database or environment (monitors behaviour)
 - logs number of transactions & response times for selected transactions via user interface
- reports based on logs, graphs of load versus response times



special considerations

expertise is required in the tool and in performance testing techniques to design tests and interpret results

34

Monitoring Tools

- continuously analyse, verify and report...
 - usage of specific system resources
 - warnings of possible service problems
- store information about software
 - helps with traceability
- often used by operators

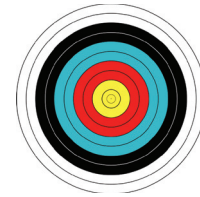


Note: this is not strictly a testing tool but provides information to assist testing in the identification and analysis of certain predefined types of defects

35

Testing of Software Structure/Architecture

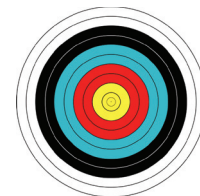
- structural: how thorough have we been?
 - a measure, usually in percentage as to how thorough our tests have been
 - more tests required to increase coverage
- may be performed at all levels
 - component: code coverage
 - integration: module coverage
 - system: menu coverage
 - acceptance: business model coverage



36

Testing Related to Changes

- changes include:
 - defect fixes
 - change requests
 - environment changes
- has the defect been fixed correctly?
 - re-testing (confirmation testing)
- has the change request been implemented correctly?
 - new and modified tests
- is the rest of the system still ok?
 - regression testing (good for automation)
- performed at all levels

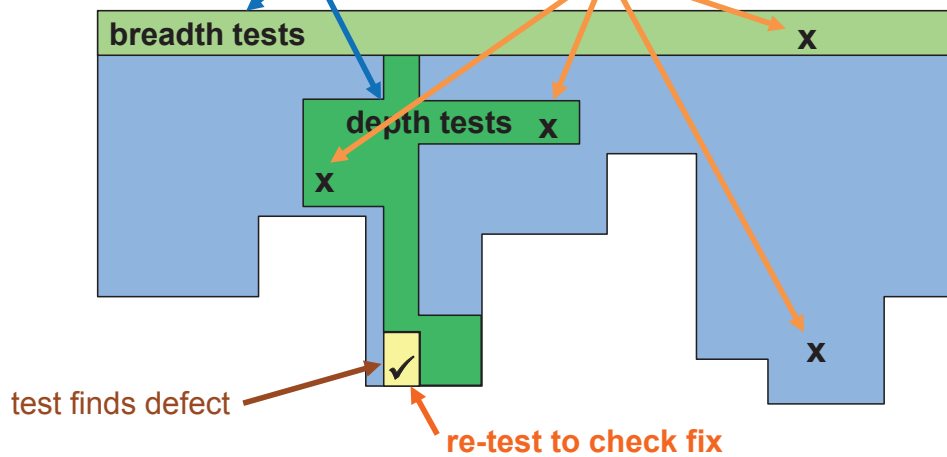


37

Re-testing vs. Regression Testing

regression tests look for unexpected side-effects (but may not find all of them)

fix introduces or uncovers new defects



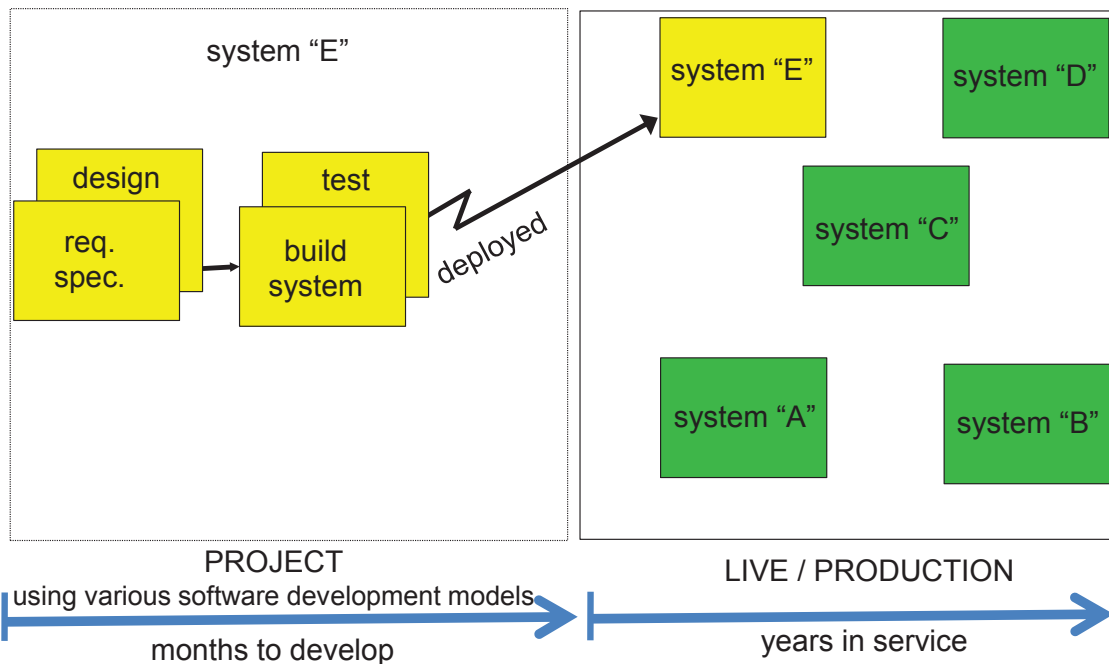
38

Contents

- 2.1 Software Development Models
- 2.2 Test Levels
- 2.3 Test Types (Targets of Testing)
- 2.4 Maintenance Testing

39

Maintenance Testing



40

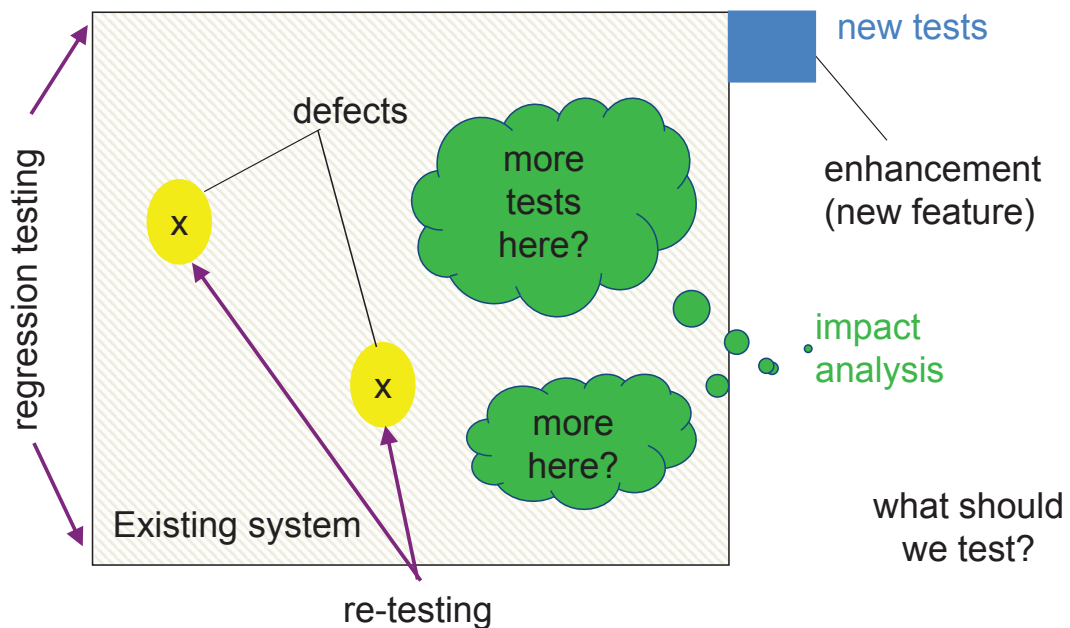
Reasons for Maintenance Testing

- modifications
 - enhancement *подобрување*, *corrective and emergency fixes*
 - environment changes, upgrades and patches
- migration (to new platforms)
 - operational test on new platform, focus on changed software
- retirement
 - data migration, archiving

I'm Retired
Not Expired

41

Small Change \neq Small Testing



42

Question: Non-functional Testing

Which of the following is an objective of non-functional testing?



- a) to measure quality characteristics of the system
- b) to check for unexpected defects in unchanged parts of the system
- c) to investigate how thoroughly the menus have been tested
- d) to check whether the system does what is in the requirements

43

Question: Match Test Targets

match the following targets

1 functional testing

2 non-functional testing

3 structural testing

4 testing of changes

with the description:



A) test code or menu structure

B) re-tests & regression tests

C) testing of quality attributes

D) tests derived from requirements

1) 1A, 2B, 3D, 4C

2) 1D, 2B, 3A, 4C

3) 1D, 2C, 3A, 4B

4) 1A, 2C, 3B, 4D

44

Summary - Key Points

- 2 software development models: sequential, iterative-incremental
- 4 test levels: component, integration, system, acceptance
- 4 types of acceptance test: user, operational, contract/regulation, alpha & beta
- 4 test types/targets: function, characteristics, structure, changes
- 7 types of characteristic testing: performance, load, stress, maintainability, reliability, portability, usability
- 3 reasons for maintenance testing: modify, migrate, retire

45

SESSION 2: TESTING THROUGHOUT THE SOFTWARE LIFECYCLE - NOTES

Terms

Commercial Off-The-Shelf (COTS), iterative-incremental development model, validation, verification, V-model.

From the ISTQB Glossary

Commercial Off-The-Shelf (COTS): A software product that is developed for the general market, i.e. for a large number of customers, and that is delivered to many customers in identical format.

validation: Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

verification: Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

V-model: A framework to describe the software development lifecycle activities from requirements specification to maintenance. The V-model illustrates how testing activities can be integrated into each phase of the software development lifecycle.

2.1 Software Development Models

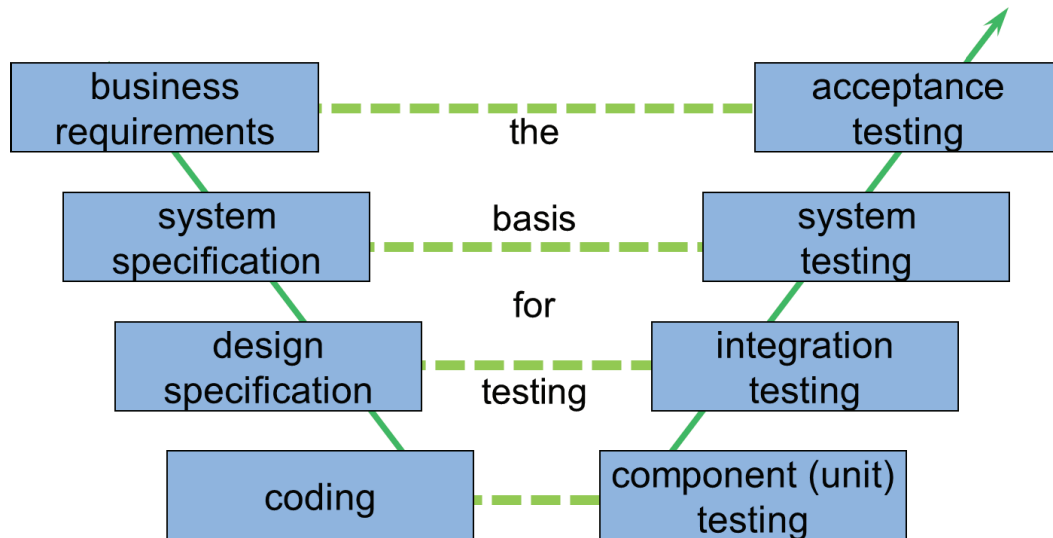
Learning Objectives:

LO-2.1.1	K2	Explain the relationship between development, test activities and work products in the development life cycle, by giving examples using project and product types.
LO-2.1.2	K1	Recognise the fact that software development models must be adapted to the context of project and product characteristics.
LO-2.1.3	K1	Recall characteristics of good testing that are applicable to any life cycle model.

2.1.1 Sequential models (V-model, W-model, Waterfall)

Structure of the V-model

In the V-model the test activities are spelled out to the same level of detail as the design activities. Software is designed and built on the left-hand (downhill) part of the model, and tested on the right-hand (uphill) part of the model. Note that different organisations may have different names for the development and testing phases. We have used the names given in the syllabus for the testing phases in our diagram.



Note that only one integration level is shown in the diagram – there are often many different levels of integration testing which are carried out in practice. For example, there may be component integration after component testing (as shown), but there may also be system integration testing after system testing, and may also be business integration testing after acceptance testing, or other levels of system integration after acceptance testing.

The correspondences between the left and right hand activities are shown by the lines across the middle of the V, showing the test levels from component testing at the bottom, integration and system testing, and acceptance testing at the top level.

However, even the V-model is often not exploited to its full potential from a testing point of view.

When are tests designed: As late as possible?

A common misconception is that tests should be designed as late as possible in the lifecycle, i.e. only just before they are needed. The reason for this is supposedly to save time and effort, and to make progress as quickly as possible. But this is progress only from a deadline point of view, not progress from a quality point of view, and the quality problems, whose seeds are sown here, will come back to haunt the product later on.

The activities on the left-hand (downhill) part of the model each produce one or more documents (typically specifications) or code. These are generally referred to as 'work products': they are the product of the work carried out within the activity identified by the boxes in the diagram. References for generic work products include Capability Maturity Model Integration (CMMI) and the standard IEEE/IEC 12207 'Software Lifecycle Processes'.

A work product is the source of what the correct results of the test should be. Even if that specification is not formally written down or fully completed, the test design activity will reveal defects in whatever work product the tests are based on. This applies to code, a part of the system, the system as a whole, or the user's view of the system. There can be work products at all levels of the V-model, from a business requirement to the code.

If test design is left until the last possible moment, then the defects are found much later when they are much more expensive to fix. In addition, the defects in the highest levels, the requirements specification, are found last - these are also the most critical and most important defects. The actual effect of this approach is the most costly and time-consuming approach to testing and software development.

When are tests designed: As early as possible!

If tests are going to be designed anyway, there is no additional effort required to move a scheduled task to a different place in the schedule.

If tests are designed as early as possible, the inevitable неизбежна effect of finding defects in the specification comes early, when those defects are cheapest to fix. In addition, the most significant defects are found first. This means that those defects are not built in to the next stage, e.g. major requirement defects are not designed in, so defects are prevented.

Testers are often under the misconception that they are constrained ограничени by the order in which software is built. The worst extreme is to have the last piece of software written be the one that is needed to start test execution. However, with test design taking place early in the lifecycle, this need not be the case.

By designing the tests early, the order in which the system should ideally be put together for testing is defined during the architectural or logical design stages. This means that the order in which software is developed can be specified before it is built. This gives the greatest opportunity for parallel testing and development activities, enabling development time scales to be minimised. This can enable total test execution schedules to be shortened and gives a more even distribution of test effort across the software development lifecycle.

An argument against this approach is that if the tests are already designed, they will need to be maintained. It is true that there will be inevitable changes due to subsequent lifecycle development stages that will affect the earlier stages. But the cost of maintaining tests must be compared with the costs of the late testing approach, not simply be accepted as negating the good points. In fact, the extent of the test design detail should be determined in part by the maintenance costs, so that less detail (but always some detail) should be designed if extensive changes are anticipated.

One of the frequent headaches in software development is a rash of requirement change requests that come from users very late in the lifecycle; a major contributing cause for this is the user acceptance test design process. When the users only begin to think about their tests just before the acceptance tests are about to start, they realise the shortcomings недостатки in the requirement specification, and request changes to it. If they had designed their tests at the same time as they were specifying those requirements, the very mental activity of test design would have identified those defects before the system had built them.

The way in which the system will be tested also serves to provide another dimension to the development; the tests form part of the specification. If you know how it will be tested, you are much more likely to build something that will pass those tests.

The end result of designing tests as early as possible is that quality is built in, costs are reduced, and time is saved in test running because fewer defects are found, giving an overall reduction in cost and effort. This is how testing activities help to build quality into the software development process.

Many of the popular Agile approaches include “test-driven development” or “test-first development” which capitalises on this effect at the developer level. Designing the tests early is one of the most widely accepted benefits of this approach.

2.1.2 Verification and validation

The Glossary definitions are:

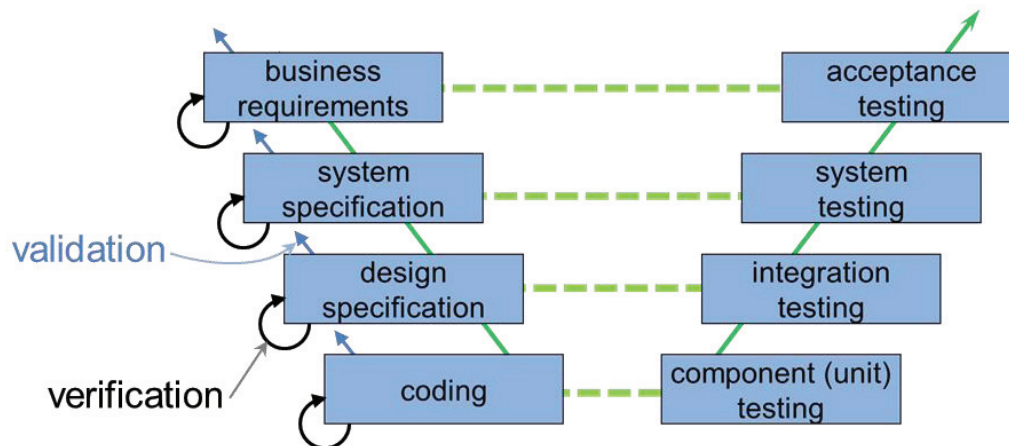
Validation: Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

Verification: Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

The key aspect to differentiate these two terms is that Validation is more concerned with use – so from a user’s perspective. Verification is more concerned with internal process.

Verification is more to do with the process of producing a work product; validation is more concerned with the products produced, i.e. the system itself.

Validating requirements may seem a little tricky given that there is probably no higher level specification. However, the validation activity is not limited to comparing one document against another. User requirements can be validated by several other means such as discussing them with end users and comparing them against your own or someone else's knowledge of the user's business and working practices. Forms of documentation (work products) other than a formal statement of requirements may be used such as contracts, memos or letters describing individual or partial requirements. Reports of surveys, market research and user group meetings may also provide a rich source of information against which a formal requirements document can be validated. In fact many of these different approaches may from time to time be applicable to the validation of any product of software development (designs, source code, etc.).



A purpose of executing tests on the system is to ensure that the delivered system has the functionality defined by the system specification. This best fits as a validation activity (since it is checking that the system has the functions that are required – i.e. that it is the right system). Verification at the system test phase is more to do with ensuring that a complete system has been built. In terms of software this is rarely a large independent task rather it is subsumed by the validation activities. However, if it were treated as an independent task it would seek to ensure that the delivered system conforms to the standards defined for all delivered systems. For example, all systems must include on-line help, display a copyright notice at start-up, conform to user interface standards, conform to product configuration standards, etc.

Many people have trouble remembering which is which, and what they both mean. Barry Boehm's definitions represent a good way to remember them: Verification is “building the system right”, and Validation is “building the right system”. Thus verification checks the correctness of the results of one development stage with respect to some pre-defined rules about what it should produce, while validation checks back against what the users really want (or what they have specified).

2.1.3 Iterative - incremental development models

In the V-model, the result of the project would be the delivery of a system, where the requirements specification would have been written first (and the tests at acceptance and system test levels), then the design documentation (and the integration tests), then the code and component tests, and then the tests would be executed from component to acceptance test, and the system would be released. It's a bit like writing a book of say 8 chapters by starting with Chapter 1, then Chapter 2, etc. A typical development may take from 6 months to 2 years.

In iterative development, the system to be built is divided into much smaller parts. One of the features is chosen to be developed first, and the requirements for that feature may be documented (and the relevant tests), then that feature would be designed, coded and tested.

After typically 1 week to 1 month, this feature or small part of the system would be complete, and could possibly be released to users (this would be called incremental *постепенно* delivery, not just incremental development). Then the next feature would be worked on (or it might be started in parallel with later work on the first increment). Assuming that the features are not standalone and independent of each other but are interconnected and interdependent, this approach is a bit like writing a bit of each chapter of the book at a time.

There are a number of different iterative-incremental approaches, as outlined below. Note that this information is provided here for additional information as it goes beyond what is required by the ISTQB Foundation syllabus (i.e. this information will not be examined in the ISTQB / BCS Certificate examination).

Prototyping (rapid prototyping)

- try something out (business or technical)
 - “running” software, not full “working” software (not production quality), intentionally incomplete
 - uses of prototypes
 - throw-away (learning what we should have built)
 - pre-production (developed further)
 - requirement basis (what the system should do)
 - test basis (compare production version)
 - experimental, evolutionary, exploratory

Rapid Application Development

- characteristics
 - users, designers, testers and developers on the team
 - product developed in time-boxes
 - number of iterations not known at start
 - development of an iteration depends on feedback from earlier iteration(s)

Rational Unified Process ®

- a software development approach originated by Rational Software Corp.
 - now a part of IBM
- a process framework / meta-model
 - designed to be tailored for each situation
- six “best practices”
 - develop software iteratively
 - manage requirements
 - component based architecture
 - visually model software
 - verify software quality
 - control changes

Agile Development Methods

- best known: Scrum, Kanban and XP (Extreme Programming).
- values:
 - communication, simplicity, feedback.
- practices:
 - planning game, small releases, metaphor, simple design, testing (tests written first and automated), refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, coding standards.

2.1.4 Good testing within a lifecycle model

There are a number of characteristics of good testing, whatever type of lifecycle development model is used.

- a corresponding test activity for every development activity (most visible in the V-model)
- each test level has corresponding objectives specific to that level
- analysis and design of tests should begin during the corresponding development activity for each level
- testers should be involved in reviewing documents as soon as possible in the lifecycle, even if they are only draft нацрт versions.

Depending on the project or context, the test levels may be different. For example, if a commercial product is purchased купени , there may be an acceptance test for that package (referred to as COTS – Commercial Off The Shelf), then there may be a system integration test with the system which the package will interact with. In this case there would not be Component Testing or System Testing.

2.2 Test Levels

Learning Objectives:

LO-2.2.1 K2 Compare the different levels of testing: major objectives, typical objects of testing, typical targets of testing (e.g., functional or structural) and related work products, people who test, types of defects and failures to be identified.

2.2.1 Component Testing

What is Component Testing?

A component is "A minimal software item that can be tested in isolation". Components are relatively small pieces of software that are, in effect the building blocks from which the system is formed. They may also be referred to as modules, units, programs, object or class and so this level of testing may also be known as module, unit or program testing. For some organisations a component can be just a few lines of source code while for others it can be a small program.

Component testing is the lowest level of test execution (i.e. it is at the bottom on the V-Model software development lifecycle). It is the opportunity to test the software in isolation and therefore in the greatest detail, looking at its functionality and structure, error handling and interfaces.

Because just a small portion of the system is being tested, it is often necessary to have a test harness or driver to form an executable program that can be executed. This will usually have

to be developed in parallel with the component or may be created by adapting a driver for another component. This should be kept as simple as possible to reduce the risk of faults in the driver obscuring замалчување faults in the component being tested. Typically, drivers need to provide a means of taking test input from the tester or a file, passing it on to the component, receiving the output from the component and presenting it to the tester for comparison with the expected outcome.

The programmer who wrote the code most often performs component testing. This is sensible because it is the most economic approach. A programmer who executes test cases on his or her own code can usually track down and fix any defects that may be revealed by the tests relatively quickly. If someone else were to execute the test cases they may have to document each failure. Eventually the programmer would come to investigate each of the incident reports, perhaps having to reproduce them in order to determine their causes. Once fixed, the fixed software would then be re-tested by this other person to confirm each defect had indeed been fixed. This amounts to more effort and yet the same outcome: defects fixed. Of course it is important that some independence is brought into the test specification activity. The programmer should not be the only person to specify test cases, as described elsewhere in this course.

Both functional and structural test case design techniques are appropriate, though the extent to which they are used should be defined during the test planning activity. This will depend on the risks involved, for example, how important, critical or complex they are.

One approach which is proving to be very effective is “test-first development” or “test-driven development”. This is one of the prime characteristics of Extreme Programming, popularised by Kent Beck in his books, and described as an agile methodology. The concept is to write the tests first, followed by the code. Given the tests are done first, and based on requirements, there is a good likelihood the subsequent code will be of much better quality.

Characteristics of component testing

The objective of component testing is to exercise the code in the greatest of detail (such as may be achieved with a thorough use of functional techniques and/or the various structural testing techniques). The following table shows the characteristics of component testing with respect to the key factors that can be identified:

Component Testing	
Objective of the test	detail of the code
Test basis	component specification, detailed design, code
Test object	components (in isolation), programs, data conversion / migration programs, database modules
Typical defects found	logic errors, boundary value errors
Tool support (general)	test management, test execution, comparison, requirements, test design, test data preparation, incident management, configuration management
Tool support (specific)	unit test frameworks, static analysis, debugging, dynamic analysis, test harness, coverage, modelling
Test approach	most thorough look at detail, white box & coverage
Responsibilities	developer (or buddy – another developer)

Tool support for component testing

The primary tool used by developers is probably what is referred to as a Unit Test Framework, for example JUnit for Java. This provides a lot of support from debugging to environment simulation.

Developers may also make use of dynamic analysis tools, which help to identify memory leaks for example. Modelling tools may also be used by developers in component testing.

A test management tool may be used to record defects though defects identified and fixed during component testing may not be formerly recorded. In some contexts recording of defects found during component testing does happen (e.g. safety critical and where component testing is performed by someone other than the author of the code). Also a test management tool may be used to record other information such as test execution progress (whether manual or automated).

Test design tools may be used to help identify the test inputs necessary to exercise specific code elements (such as a particular decision outcome or a particular combination of condition outcomes – i.e. supporting structural testing techniques). They may also help with the design of functional tests by applying common techniques such as Equivalence Partitioning and Boundary Value Analysis.

See the Student Notes for Chapter 6 for more details of these (and the other) tools.

2.2.2 Integration Testing

What is Integration Testing?

Integration testing is bringing together individual things or items that have already been tested and combining them into larger assemblies for testing now. The objective is to test that the 'set' of things function together correctly by concentrating on the interfaces between them. We are trying to find defects that couldn't be found in previous testing where the things we have now combined were tested separately. For example, although interfaces should have been tested in component testing, component integration testing investigates that the things that are communicated are correct from both sides, not just from one side of the interface.

As the scope of integration increases, e.g. as more and more components are combined together, the more difficult it becomes to isolate exactly where a defect is, when a failure occurs. This is why it is good to try to test smaller things first, and then combine things when they are more trusted. The harder it becomes to identify defects, the greater the risk that a defect, even if it is found, will not be fixed correctly.

Integration testing is concerned with testing the functionality of the interfaces, but is also concerned with testing non-functional quality characteristics if possible. For example, we could assemble a processing thread (all of the things that may happen to a specific transaction), and test just this one thread all the way through the system, before all of the exception handling is in place for events not part of the normal processing thread. This means that if we have a performance problem now, we know about it early when there is more time to do something about it.

We may also be concerned with testing the structure of the partial system that we have assembled. For example in component integration testing, we may test coverage of all calls to different objects / modules / classes / methods.

When we plan System Integration Testing there are a number of resources we might need, such as different operating systems, different machine configurations and different network configurations. These must all be thought through before the testing actually commences. We must consider what machines we will need and it might be worthwhile talking to some of the hardware manufacturers as they sometimes offer test sites with different machine configurations set up.

Characteristics of integration testing

The following table shows the characteristics of integration testing with respect to the key factors that can be identified:

Integration testing	
Objective of the test	interactions, individual features, partial system
Test basis	software and system design, architecture, workflows, use cases
Test object	sub-systems database implementation, infrastructure, interfaces, system configuration, configuration data
Typical defects found	missing parameters in communication between components or systems
Tool support (general)	test management, test execution, comparison, requirements, test design, test data preparation, incident management, configuration management
Tool support (specific)	test harness, modelling, coverage, dynamic analysis, monitoring
Test approach	incremental integration: top-down, bottom-up, minimum capability, thread
Responsibilities	developers (sometimes an independent team)

With any level of integration, it is best to take an incremental approach rather than a “big bang” approach. This means identifying how many things will be combined in one step, and in what order. This is known as an “integration strategy”. It can be based on architecture (top-down, bottom-up or others), functional tasks, processing sequence or anything else that makes sense.

“Big Bang” integration means putting together all of the components in one go. The philosophy is that we have already tested all of the things to be combined so why not just throw them all in together and test the lot? The reason normally given for this approach is that it saves time - or does it? If we encounter a problem it tends to be harder to locate and fix the defects. If the defect is found and fixed then re-testing (confirmation testing) usually takes a lot longer. In the end the Big Bang strategy does not work - it actually takes longer this way. This approach is based on the [mistaken] assumption that there will be no defects. So whilst it is still an approach it is not one we would recommend.

Incremental integration is where a small number of components are combined at once. At a minimum, only one new thing (component, system) would be added at each integration step. This has the advantage of much easier defect location and fixing, as well as faster and easier recovery if things do go badly wrong. (The finger of suspicion would point to the most recent addition to the growing integration.)

If the order of integration (at whatever level) is known early, then the integration order can be used to “drive” the order of how the things to be combined are built. This gives more efficient testing because the things needed for the first integration can be built first. Then the integration testing of those things can start while other things are still being built. This can save elapsed time, which is always welcome in testing! Please note that it does not save effort as the items still need to be tested anyway.

2.2.3 System Testing

What is System Testing?

System testing is the testing of a system, which is “a collection of components organised to accomplish a specific function or set of functions”. A system is normally something that is produced by a project, but a system in one organisation may be a lot bigger than a system in another organisation.

System testing focuses on the functionality of the whole system, e.g. end to end of whatever is within the scope of this system. System testing also tests non-functional aspects, i.e. quality

characteristics or attributes. The non-functional aspects are often as important as the functional, but are generally less well specified and may therefore be more difficult to test (but not impossible). System testing may also be based on structure, but not code-based structures. Coverage of things such as menu options, screens, use cases or most important user transactions would be appropriate structural items at system level.

If an organisation has an independent test group, it is usually at this level, i.e. it performs system testing.

Note that we are still looking for defects in system testing, this time in end-to-end functionality and in things that the system as a whole can do that could not be done by only a partial baseline.

Characteristics of system testing

The following table shows the characteristics of system testing with respect to the key factors that can be identified:

System testing	
Objective of the test	end to end functionality and quality characteristics
Test basis	system and software requirement specification, use cases, functional specification, risk analysis reports
Test object	system, user and operation manuals, system configuration, configuration data
Typical defects found	incorrect function, quality characteristics (e.g. performance)
Tool support (general)	test management, test execution, comparison, requirements, test design, test data preparation, incident management, configuration management
Tool support (specific)	performance, security, monitoring
Test approach	realistic environment, black box, coverage of system level items
Responsibilities	independent team (sometimes outsourced)

Tools for System Testing

There are many tools available for the system testing function to utilise, however the primary tool when running the tests will be a test execution tools supplemented with comparison tools and incident management tools.

See the Student Notes for Chapter 6 for more details of these (and the other) tools.

2.2.4 Acceptance Testing

What is Acceptance Testing?

Acceptance testing is the responsibility of the ultimate users of the system, i.e. the customers and stakeholders for the system.

Acceptance testing is where the focus or objective of the testing changes. In the earlier levels, the main objective (usually) is to find as many defects as possible, by causing failures. In acceptance testing, our primary aim is to gain confidence that the system will be suitable for real use by real users. This difference in objective is the key difference between system and acceptance testing, since acceptance testing is also testing of a system.

Although acceptance testing is covered in the syllabus after integration and then system testing, in practice there may be some acceptance testing done very early on (e.g. of a package component), or there may be large-scale integration testing done after acceptance testing of a system.

There are a number of types of acceptance testing listed in the syllabus:

User acceptance testing

User Acceptance Testing is the final stage of validation. This is the time that customers get their hands on the system (or should do) and the end product of this is usually a sign-off from the users.

One of the problems is that this is rather late in the project for users to be involved - any problems found now are too late to do anything about them. This is one reason why Rapid Application Development (RAD) has become popular - users are involved earlier and testing is done earlier.

However, the users should have been involved in the test specification of the Acceptance Tests at the start of the project. They should also have been involved in reviews throughout the project, and there is nothing to say that they cannot be involved in helping to design System and Integration tests. So there really should be no surprises!

Operational (acceptance) testing

Operational staff, system administrators or database administrators may have special requirements for testing, including:

- testing of backup functions, and particular restoring from backups;
- disaster recovery, which includes restoring from backups, but also transferring day to day running to a disaster recovery site;
- user management, such as password and access control;
- maintenance tasks for fixes or enhancements;
- data load and migration tasks;
- security aspects, including firewalls, looking for new vulnerabilities and virus checking at regular intervals.

Contract and regulation acceptance testing

If a system is the subject of a legally binding contract, there may be aspects directly related to the contract that need to be tested. It is important to ensure that the contractual documents are kept up to date; otherwise you may be in breach of a contract while delivering what the users want (instead of what they specified two years ago). However, it is not fair for users to expect that the contract can be ignored, so the testing must be against the contract and any agreed changes.

An industry that is subject to legal, governmental or safety regulations will need to ensure that they test to meet those regulations.

Alpha and beta testing

Both alpha and beta testing are normally used by software houses that produce mass-market shrink-wrapped software packages. This stage of testing is normally after system testing; but it may include elements of system integration testing particularly with different environmental factors or platforms. The alpha or beta testers are given a pre-release version of the software and are asked to give feedback on the product. Alpha and beta testing is done where there are no identifiable "end users" other than the general public.

The difference between alpha and beta testing is where they are carried out. Alpha testing is done on the development site - potential customers would be invited in to the developers' offices. Beta testing is done on customer sites - the software is sent out to them to test in their own offices (or at home).

Other terms that may be used for acceptance testing include "factory acceptance testing" for testing before a system is sent to a customer site, and "site acceptance testing" after the system is moved. This is also important for the hardware configuration, especially if it is unique to that customer.

Characteristics of acceptance testing

The following table shows the characteristics of acceptance testing with respect to the key factors that can be identified:

Acceptance testing	
Objective of the test	business view of system, business procedures
Test basis	user requirements, system requirements, use cases, business processes, risk analysis reports
Test object	business processes on fully integrated system, operational and maintenance processes, user procedures, forms, reports, configuration data
Typical defects found	mis-match with business needs, misunderstanding of business processes
Tool support (general)	test management, test execution, comparison, requirements, test design, test data preparation, incident management, configuration management
Tool support (specific)	performance, security, monitoring
Test approach	realistic environment, black box based on user profiles, coverage of business scenarios
Responsibilities	independent team of users or customers

The slide “tailoring of test levels” shows where typically they apply to systems/applications. However, these are examples only and the final decision depends on the context of the application.

2.3 Test Types (Targets of Testing)

Learning Objectives

LO-2.3.1	K2	Compare four software test types (functional, non-functional, structural and change-related) by example.
LO-2.3.2	K1	Recognise that functional and structural tests occur at any test level.
LO-2.3.3	K2	Identify and describe non-functional test types based on non-functional requirements.
LO-2.3.4	K2	Identify and describe test types based on the analysis of a software system's structure or architecture.
LO-2.3.5	K2	Describe the purpose of confirmation testing and regression testing.

There are four types of objectives for testing; these are described as test types or targets of testing. Depending on what the objectives are, different test efforts will be organised differently. For example, system testing aimed at testing performance would be quite different to component testing aimed at achieving decision coverage.

Models of the software may be developed and/or used in functional testing (e.g. a process flow model, a state transition model or a plain language specification), non-functional testing (e.g. performance model, usability model, security threat modelling), and structural testing (e.g., a control flow model or menu structure model).

2.3.1 Testing of function (functional testing)

The function of a system (or component) is “what it does”. This is typically described in a requirements specification or a functional specification, or in use cases. There may be some functions that are “assumed” to be provided that are not documented – these also form part of the requirement for a system, though it is difficult to test against undocumented requirements!

As an example, we can look at testing functionality from two perspectives, requirements based or business process based.

Requirements-based testing uses a specification of the functional requirements for the system as the basis for designing tests. A good way to start is to use the table of contents of the requirement specification as an initial test inventory or list of items to test (or not to test). We should also prioritise the requirements based on risk criteria (if this is not already done in the specification) and use this to prioritise the tests. This will ensure that the most important and most critical tests are included in the system testing effort.

Business process-based testing uses knowledge of the business profiles (or expected business profiles). Business profiles describe the birth to death situations involved in the day to day business use of the system. For example, a personnel and payroll system may have a business profile along the lines of: someone joins company, he or she is paid on a regular basis, he or she leaves the company.

Another business process-based view is given by user profiles. User profiles describe how much time users spend in different parts of the system. For example, consider да се разгледа a simple bank system that has just three functions: account maintenance, account queries and report generation. Users of this system might spend 50% of their time using this system performing account queries, 40% of their time performing account maintenance and 10% of their time generating reports. User profile testing would require that 50% of the testing effort is spent testing account queries, 40% is spent testing account maintenance and 10% is spent testing report generation.

Use cases are popular in object-oriented development. These are not the same as test cases, since they tend to be a bit “woolly” but they form a useful basis for test cases from a business perspective.

The techniques used for functional testing are often specification-based, but intuitive techniques can also be used. For example security testing may be done against a specification for a firewall, or a description of vulnerabilities слабости.

Tools for Security Testing

Security testing tools are covered here since they cover the testing of security functions. This type of tool could be used at any test level, from component testing through acceptance testing.

See Chapter 6 for more detail about this and other tools.

2.3.2 Testing of non-functional software characteristics (non-functional testing)

A second type of testing is the testing of the quality characteristics, or non-functional characteristics of software under test. Here we are interested in how well or how fast something is done. We are testing something that we can measure on a scale of measurement, for example time to respond.

Non-functional testing, as functional testing, can be performed at all test levels. It considers the external behaviour of the software and in most cases uses test cases derived from black-box test design techniques to accomplish that.

There are two standards that describe a set of quality characteristics (or quality attributes). The first of these, ISO 9126: Software Product Quality, is referenced by the syllabus. The

second actually supersedes the first (it was published after the syllabus): ISO/IEC 25000 Systems and software Quality Requirements and Evaluation (SQuaRE). The quality characteristics are:

- Functionality
- Usability
- Efficiency
- Reliability
- Portability
- Maintainability

Note that the first of these, Functionality, is not a non-functional characteristic as it is about the functionality (a good quality system must have the correct functionality after all ☺). The standards divide each of these into a number of sub-characteristics to focus on specific aspects (e.g. efficiency is divided into performance and resource use). The Foundation syllabus highlights нагласува a subset of the quality characteristics but the ISTQB advanced level

courses look at them all in more detail.

Usability testing

Testing for usability is very important, but cannot be done well by technical people; it needs to have input from real users.

Usability design and testing is a rich field in its own right, with many very good books and interesting web sites.

Testers are sometimes the “last line of defence” for the users when usability issues arise. Technical people may not be aware of the difficulty of use of some aspects because they are so computer-literate themselves. We need to give this area due consideration given the legislation there is in the UK and other countries that these aspects are dealt with.

Performance, load & stress testing

Performance tests include timing tests such as measuring response times to a PC over a network, but may also include response times for performing a database back-up for example.

Load tests, or capacity or volume tests are test designed to ensure that the system can handle what has been specified, in terms of processing throughput, number of terminals connected, etc. Stress tests see what happens if we go beyond those limits.

Some systems have specific storage objectives, for example, that limit the amount of computer memory that may be used for a given set of operations or the amount of disc space that may be used for temporary file storage. Storage testing is about determining whether or not these storage objectives are met. Embedded software is more likely to have specific storage objectives.

Performance testing tools provide facilities to generate loads on a system to probe its behaviour. Monitoring tools, although not strictly speaking a testing tool, can provide much useful information about the system as it is being tested, as well as when it is operational.

More details about these and other tools can be found in Chapter 6.

Reliability testing

If a specification says “the system will be reliable”, this statement is untestable. Qualities such as reliability, maintainability, portability, availability etc. need to be expressed in measurable terms in order to be testable. Mean Time Between Failures (MTBF) is one way of quantifying reliability.

Portability testing

There can be many different aspects to consider here. Different users may have different hardware configurations such as amount of memory; they may have different software as well, such as word processor versions. If the system is supposed to work in different

configurations, it must be tested in all or at least a representative set of configurations. For example, web sites should be tested with different browsers.

Upgrade paths also need to be tested; sometimes an upgrade of one part of the system can be in conflict with other parts.

How will the new system or software be installed on user sites? The distribution mechanism should be tested. The final intended environment may even have physical characteristics that can influence the working of the system.

Maintainability testing

We can test for how maintainable a system or component is, by measuring aspects of sample changes made to the system, for example how long it takes to make a change, how helpful the system documentation is, etc. This is a test of the maintenance documentation.

There are other aspects of non-functional testing that are not specifically mentioned in the syllabus, such as backup and recovery testing.

2.3.3 Testing of software structure and coverage tools (structural testing)

The third type of testing is the structure of the system or component. If we are talking about the structure of a system, we may call it the system architecture.

Structural testing is often referred to as “white box” or “glass box” because we are interested in what is happening “inside the box”. It is most often used as a way of measuring the thoroughness of testing through the coverage of a set of structural elements or coverage items. Structural testing can occur on any test level.

At component level, and to a lesser extent at component integration testing, there is good tool support to measure code coverage. Coverage measurement tools assess the percentage of executable elements (e.g. statements or decision outcomes) that have been exercised (i.e. covered) by a set of test cases. If coverage is not 100%, then additional tests may be written and run to cover those parts that have not yet been exercised.

2.3.4 Testing related to changes (re-testing and regression testing)

The final type of testing is the testing of changes. This category is slightly different to the others because if you have made a change to the software, you will have changed the way it functions, the way it performs (or both) and its structure. However we are looking here at the specific types of tests relating to changes, even though they may include all of the other test types.

Re-testing

When a test fails and we determine the cause of the failure is a software defect, the defect is reported, we can expect a new version of the software that has had the defect fixed. In this case we will need to execute the test again to confirm that the defect has indeed been fixed. This is known as re-testing (also known as confirmation testing).

When doing re-testing, it is important to ensure that the test is executed in exactly the same way as it was the first time, using the same inputs, data and environment. If the test now passes does this mean that the software is now correct? Well, we now know that at least one part of the software is correct – where the defect was. But this is not enough. The fix may have introduced a new or uncovered defect elsewhere in the software. The way to detect these “unexpected side-effects” of fixes is to do regression testing.

Regression testing

Like re-testing testing, regression testing involves executing test cases that have been executed before. The difference is that for regression testing the test cases probably passed the last time they were executed (compare this with the test cases executed in re-testing - they failed the last time).

The purpose of regression testing is to verify that modifications in the software or the environment have not caused unintended adverse негативни side effects and that the system still meets its original requirements.

It is common for organisations to have what is usually called a regression test suite naker or regression test pack. This is a set of test cases that is specifically used for regression testing. They are designed to collectively exercise most functions (certainly the most important ones) in a system but not test any one in detail. It is appropriate to have a regression test suite at every level of testing (component testing, integration testing, system testing, etc.). All of the test cases in a regression test suite would be executed every time a new version of software is produced and this makes them ideal candidates for automation. If the regression test suite is very large it may be more appropriate to select a subset for execution.

Regression tests are executed whenever the software changes, either as a result of fixes or new or changed functionality. It is also a good idea to execute them when some aspect of the environment changes, for example when a new version of a database management system is introduced or a new version of a source code compiler is used.

Maintenance of a regression test suite should be carried out so it evolves over time in line with the software. As new functionality is added to a system new regression tests should be added and as old functionality is changed or removed so too should regression tests be changed or removed. As new tests are added a regression test suite may become very large. If all the tests have to be executed manually it may not be possible to execute them all every time the regression suite is used. In this case a subset of the test cases has to be chosen. This selection should be made in light of the latest changes that have been made to the software. Sometimes a regression test suite of automated tests can become so large that it is not always possible to execute them all. It may be possible and desirable to eliminate some test cases from a large regression test suite for example if they are repetitive (tests which exercise the same conditions) or can be combined (if they are always run together). Another approach is to eliminate test cases that have not found a defect for a long time (though this approach should be used with some care!).

2.4 Maintenance Testing

Learning Objectives

- | | | |
|----------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LO-2.4.1 | K2 | Compare maintenance testing (testing an existing system) to testing a new application with respect to test types, triggers for testing and amount of testing. |
| LO-2.4.2 | K1 | Recognise indicators for maintenance testing (modification, migration and retirement). |
| LO-2.4.3 | K2 | Describe the role of regression testing and impact analysis in maintenance. |
-

Maintenance Testing is all about preserving the quality we have already achieved. Once the system is operational, then any further enhancements or defect fixes will be part of the on-going maintenance of that system – the testing done for these changes to an existing working system is maintenance testing.

There are three types of things that can trigger changes to an existing system: modifications, migration or retirement.

Modifications include defect fixes and planned enhancements, operating system, database and COTS (Commercial Off The Shelf) package upgrades. Also the change of the underlying infrastructure such as middleware or servers, or even upgrades to the software may be a good reason to do some maintenance testing.

If a system is to be moved from one environment or platform to another, this is called migration of the system, and this will require testing. Migration testing (conversion testing) is also needed when data from another application will be migrated into the system being maintained.

When a system is to be phased out of use, i.e. retired, then data may need to be transferred to another system or archived. These also need to be tested. Another consideration for a retired system is how easy is it to retrieve it from an archive. E.g. if an accounts package is changed, does the new package allow you to access old data in case it's required.

Because the system is already there, when something is changed, there is a lot of the system that should still work, so maintenance testing involves a lot of regression testing as well as the testing of the changes.

It is worth noting that there is a different sequence with Maintenance Testing. In development we start from small components and work up to the full system; in maintenance testing, we can start from the top with the whole system. This means that we can make sure that there is no effect on the whole system before testing the individual fix. We also have different data - there is live data available in maintenance testing, whereas in development testing we had to build the test data.

A breadth test is a shallow but broad test over the whole system, often used as a regression suite. Depth tests explore specific areas such as changes and fixes. Impact analysis investigates the likely effects of changes, so that the testing can be deeper in the riskier areas.

It is often argued that maintenance testing is the hardest type of testing to do because:

- there are no specifications;
- any documentation is out-of-date;
- lack of regression test scripts;
- domain knowledge is limited due to age of the system (and programmers!).

If you do not have good specifications, it can be argued that you cannot test. The specification is the oracle that tells the tester what the system should do.

So what do we do? Although this is a difficult situation, it is very common, and there are ways to deal with it. Make contact with those who know the system, i.e. the users. Find out from them what the system does do, if not what it should do. Anything that you do learn: document. Document your assumptions as well so that other people have a better place to start than you did. Track what it is costing the company in not having good, well maintained specs

To find out what the system should do, you will need some form of oracle. This could be the way the system works now. Another suggestion is to look in user manuals or guides (if they exist). Finally, you may need to go back to the experts and "pick their brains".

You can validate what is already there but not verify it (nothing to verify against).

2.5 Summary

The following table shows the characteristics of each level of testing with respect to the key factors that can be identified:

	Component	Integration	System	Acceptance
Objective of the test	component spec, detailed design, code	interactions, individual features, partial system	end to end, quality characteristics	business view of system, business procedures
Test basis	components (in isolation), programs, data conversion / migration programs, database modules	software and system design, architecture, workflows, use cases	sub-systems database implementation, infrastructure, interfaces, system configuration, configuration data	system, user and operation manuals, system configuration, configuration data
Test object	code, component in isolation	sub-systems database implementation, infrastructure, interfaces, system configuration, configuration data	system, user and operation manuals, system configuration, configuration data	business processes on fully integrated system, operational and maintenance processes, user procedures, forms, reports, configuration data
Typical defects found	logic errors, boundary value errors	missing parameters in communication between components or systems	incorrect function, quality characteristics (e.g. performance)	mis-match with business needs, misunderstanding of business processes
Tool support (general)	test mgt, test execution. comparison, requirements, test design, test data preparation, incident mgt, config mgt	test mgt, test execution. comparison, requirements, test design, test data preparation, incident mgt, config mgt	test mgt, test execution. comparison, requirements, test design, test data preparation, incident mgt, config mgt	test mgt, test execution. comparison, requirements, test design, test data preparation, incident mgt, config mgt
Tool support (specific)	unit test frameworks, static analysis, debugging, dynamic analysis, test harness, coverage, modelling	test harness, modelling, coverage, dynamic analysis, monitoring	performance, security, monitoring	performance, security, monitoring
Test approach	most thorough look at detail, white box & coverage	incremental integration: top-down, bottom-up, minimum capability, thread	realistic environment, black box, coverage of system level items	realistic environment, black box based on user profiles, coverage of business scenarios
Responsibilities	developer (or buddy – another developer)	developers (sometimes an independent team)	independent team (sometimes outsourced)	independent team of users or customers

Session 3

Static Techniques

Contents

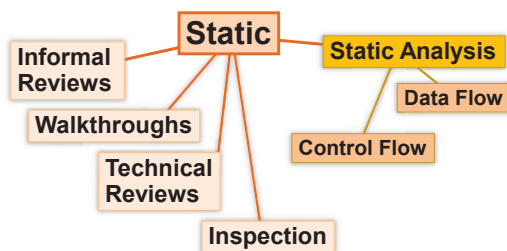
3.1 Static Techniques and the Test Process

3.2 The Review Process

3.3 Static Analysis

2

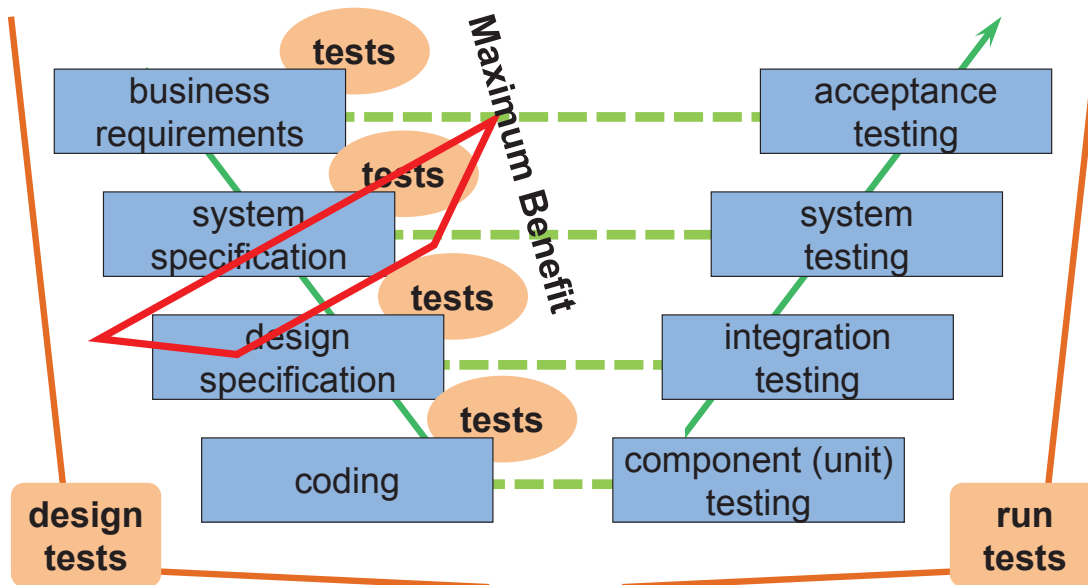
Some Static Test Techniques



- find defects
- do not execute the code
- manual (reviews) and automated (static analysis)
- should be performed before dynamic testing for maximum benefit
- different static test techniques will find different defects

3

What and When to Review?



4

Example Document Defects

- from a non-disclosure agreement from a client:
 - “Both parties shall exercise a reasonable degree of care in the handling and storage of the confidential information so as to ensure its inadvertent disclosure to unauthorised persons.”
- from a speaker contract:
 - “This letter sets forth the agreement between you (speaker) and <conference organiser> ...
 - ... If you are providing the presentation as part of your duties with your company or another company, please let me know and have a speakerized representative of the company also sign this agreement.”

5

Benefits of Reviews

- early defect detection and correction
- development productivity improvement
- reduced development timescales
- reduced testing time and cost
- lifetime cost reductions
- reduced defect levels
- improved customer relations and communication
- find omissions in requirements



6

Requirements Management Tools

- automated support for verification and validation of requirements models
 - stores requirement statements
 - ▶ checks for consistency
 - ▶ helps prioritise
 - traceability to design, code, tests
 - ▶ helps manage other documentation
 - ▶ duplicates some test management tool functionality
 - ▶ reports coverage of requirements, functions and features by the tests



7

Contents

3.1 Static Techniques and the Test Process

3.2 The Review Process

3.3 Static Analysis

8

Varying Formality of Reviews

very informal



very formal

level of formality depends on:

- maturity of the development process
- legal or regulatory requirements
- the need for an audit trail
- time versus quality

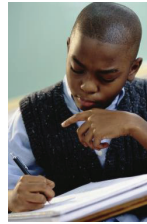
objectives of the review

- find defects
- gain understanding
- discussion
- decision making by consensus

9

Types of Review

- informal review
 - inexpensive, some benefit
- walkthrough
 - learning, understanding, find defects
- technical review
 - discuss, make decisions, evaluate alternatives, find defects, solve technical problems, check standards
- inspection
 - most formal, find defects

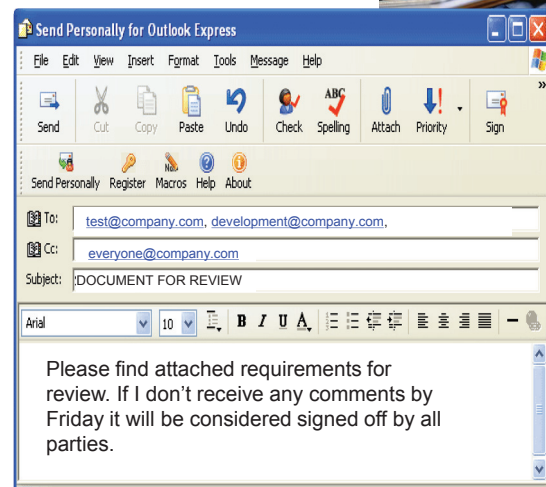


these three types can be peer reviews (i.e. conducted by people only at the same hierarchical level)

10

Informal Review

- characteristics
 - no formal process (normally undocumented)
 - widely viewed as useful & cheap (but hard to prove)
 - a helpful first step for any organisation
 - document issued with "please comment"



11

Walkthrough

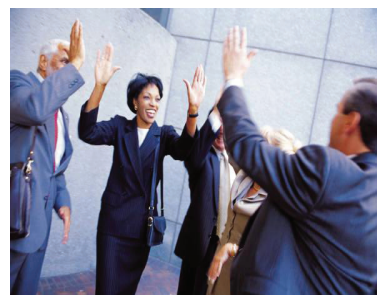
- characteristics
 - author guides the group through a document and his/her thought processes
 - may vary from informal to very formal
 - optional pre-meeting preparation
 - can be a pre-requisite to other types of review
 - consensus often reached



12

Technical Review

- characteristics
 - clearly defined defect-detection process - normally documented
 - includes peer & technical experts (peer reviews without management)
 - ideally led by trained moderator
 - optional use of checklists
 - can be rather subjective
 - may vary from informal to very formal



13

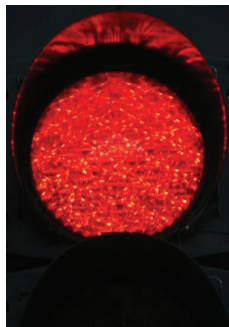
Inspection

- characteristics
 - documents need to be written down
 - led by trained moderator with defined roles
 - formal process based on rules, checklists, entry/exit criteria
 - includes pre-meeting preparation and formal follow-up
 - usually peer examination
 - source documents issued
 - change requests to source documents may be issued
 - includes gathering metrics



14

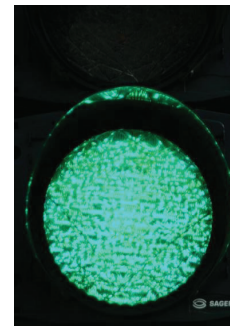
Outcome of Reviews



product has so many issues that need resolving - further reviews are necessary



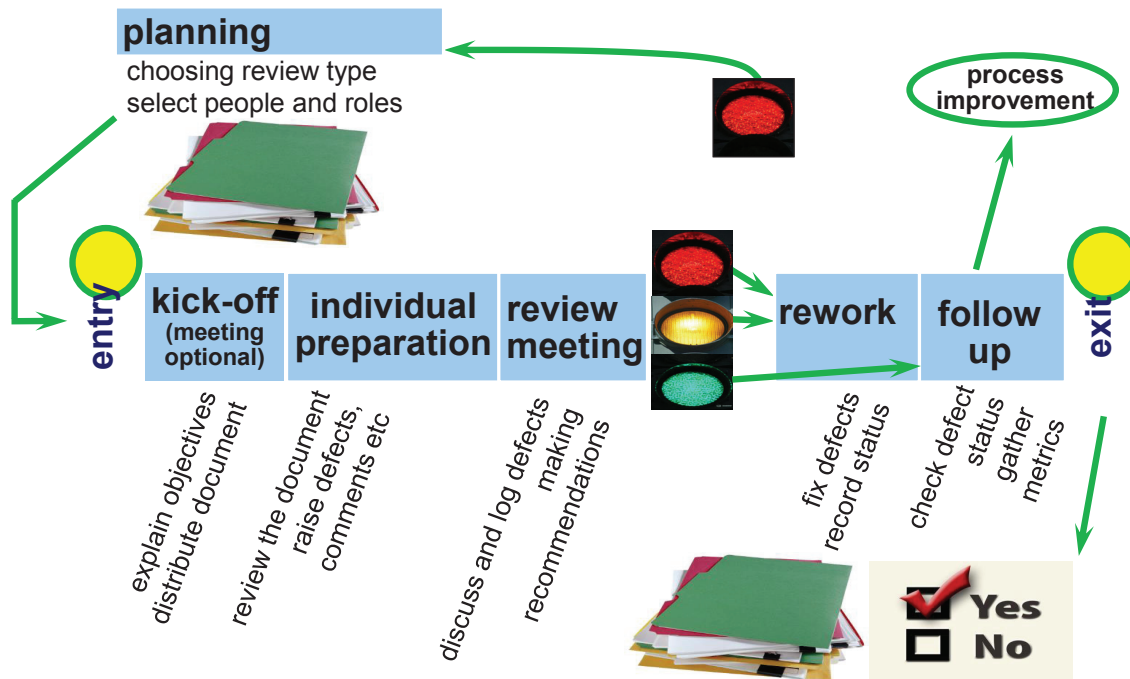
product has issues that need resolving, but no further reviews are required



product can be used in its current form – no further reviews are required

15

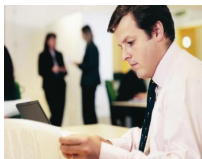
Phases of a Formal Review



16

Roles and Responsibilities

manager



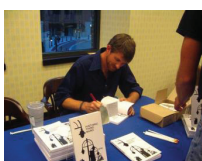
- decides on review type
- allocates time
- establishes whether objectives met

moderator



- leads the review
- chairs the meeting
- mediates when disagreements occur

author



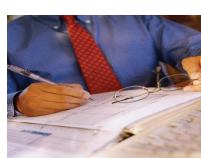
- chief responsibility for the document
- correcting the document
- answering questions to the document

reviewers



- identify defects, raise comments & questions
- specifically chosen
- attend meetings

scribe



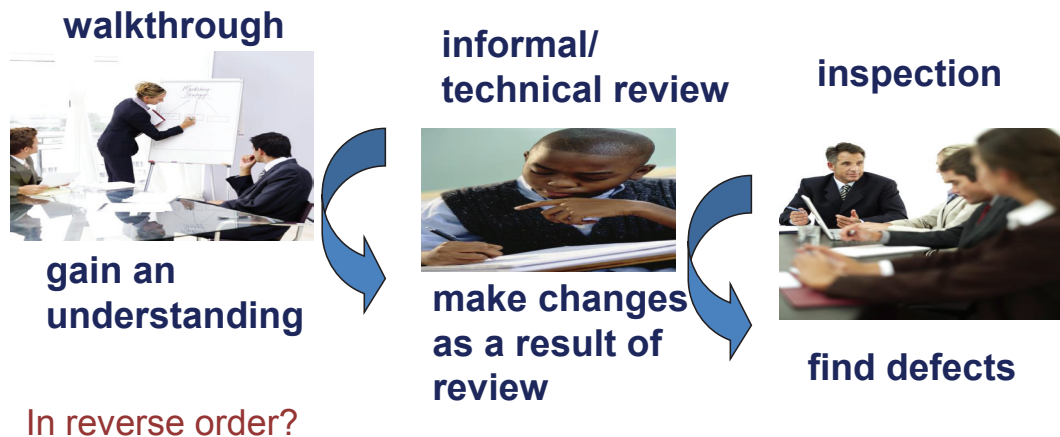
- note taker or recorder
- documents defects, comments & questions



17

Employing More Than one Type

- consider having more than one type of review for the same document
 - at different stages of its development



18

Why Reviews Don't Always Work

- lack of training in the technique
 - particularly Inspection (most formal), right people not involved
- lack of or quality of documentation
 - what is being reviewed / Inspected
- lack of management support - "lip service"
 - want them done, but don't allow time for them
- failure to improve processes
 - disheartening to repeatedly find the same defects
- people issues & psychological aspects not dealt with
 - subjective/negative experience, defects are not welcome



19

Success Factors for Reviews



- review objectives defined
- the right review is applied to achieve the objective
- right people involved
- training in reviews is given
- effectiveness of the review is enhanced (checklists & roles)
- testers (and others) are valued members of the review team
- defects are welcome (when expressed objectively)
- people issues understood and managed
- reviews conducted in an atmosphere of trust
- management provide support (appropriate time)
- process improvement built-in to the review process



20

Review Tools (Review Process Support Tools)

- mostly “in-house” built
 - spreadsheets, databases and templates
- what can they do?
 - stores information about the review process
 - ▶ comments, defects, effort, changes, metrics
 - provides traceability between
 - ▶ rules and checklists
 - ▶ documents and source code
 - helps with on-line reviews
 - ▶ infrastructure if teams are geographically dispersed



21

Contents

3.1 Static Techniques and the Test Process

3.2 The Review Process

3.3 Static Analysis

22

What is Static Analysis?

program code/HTML/XML/etc.

```
# Remember the number of words in the list.
$noWords = @wordList;

# Place first word.
$word = shift(@wordList);
placeWordAcross($word, length($word), 0, 0);

# While words have been added.
while ($noWords > @wordList)
{
    # Update count of words.
    $noWords = @wordList;

    # While there remains an untried word.
    $curWord = 0;
    while ($curWord < @wordList)
    {
        # If current word can be placed.
        $wordLength = length($wordList[$curWord]);
        if (placeNextWord($wordList[$curWord], $wordLen))
        {
            # Remove this word from the list.
            splice (@wordList, $curWord, 1);
        }
        else # Word not placed.
        {
            # Move on to next word.
            $curWord++;
        }
    }
}
```

- a form of automated testing
 - checks for violations of standards
 - checks for things that may be wrong
 - finds defects rather than failures
- extends compiler technology
 - compilers statically analyse code

static analysis
performs checks and reports on defects and anomalies

report

Use of uninitialised value in length at line 6.
Name "wordLength" used only once at line 19.
Name "wordLen" used only once at line 20.
Name "cutWord" used only once at line 23.

remember

static techniques do not execute the code

23

The Benefits of Static Analysis

note

static analysis is usually performed by a tool

- defect detection
 - finding defects early prior to execution - by developers during coding, and by designers during software modelling
 - identifying defects not easily found during dynamic testing
- defect prevention
 - lessons learned in development prevents defects in future projects
 - improved maintainability of the code
- early warning signs
 - suspicious aspects of the code (e.g. code complexity is high)
 - detecting inconsistencies and dependencies in software models (e.g. links)

24

Static Analysis Tools (Developer)

- provide information about the quality of software
 - code is examined, not executed
 - helps enforce coding standards and aids understanding of the code
- supports developers, testers and QA teams
 - finds defects before test execution, saves time & money
 - may accelerate and improve process by having less rework
- supports managers
 - objective measures:
e.g. CC* & LOC*
(for risk analysis)



special considerations

may generate lots of messages, warning messages need addressing, filtering of messages is desirable

* CC: cyclomatic complexity
LOC: lines of code

25

Typical Defects Found by Static Analysis Tools

- referencing a variable with an undefined value
- variables that are never used
- programming standards violations
- inconsistent interfaces between modules
- potential infinite loops
- unreachable (dead) code
- security vulnerabilities
- syntax violations of code and software models



26

Examples: Data Flow & Control Flow Defects

What is wrong with this code?

```
n = 0
read (x)
n = 1
while x > y do
    read (y)
    write (n*y)
    x = x - n
end while
if n = 2 then
    write (n*x)
endif
```

n is re-defined without being used

y is used before it has been defined (first time around the loop)

dead code n is set to 1 and never changed

27

Limitations and Advantages

- limitations:
 - cannot distinguish "fail-safe" code from programming defects
 - often creates large numbers of warning messages that need to be managed
 - does not execute the code, so not related to operating conditions
- advantages:
 - can find defects difficult to "see"
 - gives objective quality assessment of code (metrics)

28

Summary – Key Points

- reviews: why & when? find defects,
as early as possible = cost saving
- what to review? any development and test documents, code
- types of review: Informal review, Walkthrough,
Technical review, Inspection
- static analysis: main characteristic: doesn't execute the code
- static analysis provides information on: defects,
quality of the code & code metrics

29

SESSION 3: STATIC TECHNIQUES - NOTES

Terms

compiler, complexity, control flow, data flow, dynamic testing, entry criteria, formal review, informal review, inspection, metric, moderator, peer review, reviewer, scribe, static analysis, static testing, technical review, walkthrough.

From the ISTQB Glossary

compiler: A software tool that translates programs expressed in a high-order language into their machine language equivalents.

complexity: The degree to which a component or system has a design and/or internal structure that is difficult to understand, maintain and verify.

control flow: A sequence of events (paths) in the execution through a component or system.

data flow: An abstract representation of the sequence and possible changes of the state of data objects, where the state of an object is any of creation, usage, or destruction.

dynamic testing: Testing that involves the execution of the software of a component or system.

entry criteria: The set of generic and specific conditions for permitting a process to go forward with a defined task, e.g., test phase. The purpose of entry criteria is to prevent a task from starting which would entail more (wasted) effort compared to the effort needed to remove the failed entry criteria.

formal review: A review characterized by documented procedures and requirements, e.g., inspection.

informal review: A review not based on a formal (documented) procedure.

inspection: A type of peer review that relies on visual examination of documents to detect defects, e.g., violations of development standards and non-conformance to higher level documentation. The most formal review technique and therefore always based on a documented procedure.

metric: A measurement scale and the method used for measurement.

moderator: The leader and main person responsible for an inspection or other review process.

peer review: A review of a software work product by colleagues of the producer of the product for the purpose of identifying defects and improvements. Examples are inspection, technical review and walkthrough.

reviewer: The person involved in the review that identifies and describes anomalies in the product or project under review. Reviewers can be chosen to represent different viewpoints and roles in the review process.

scribe: The person who records each defect mentioned and any suggestions for process improvement during a review meeting, on a logging form. The scribe should ensure that the logging form is readable and understandable.

static analysis: Analysis of software development artefacts, e.g., requirements or code, carried out without execution of these software development artefacts. Static analysis is usually carried out by means of a supporting tool.

static testing: Testing of a software development artefact, e.g., requirements, design or code, without execution of these artefacts, e.g., reviews or static analysis.

technical review: A peer group discussion activity that focuses on achieving consensus on the technical approach to be taken.

walkthrough: A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content.

3.1 Static Techniques and the Test Process

Learning Objectives

LO-3.1.1	K1	Recognise software work products that can be examined by the different static techniques.
LO-3.1.2	K2	Describe the importance and value of considering static techniques for the assessment of software work products.
LO-3.1.3	K2	Explain the difference between static and dynamic techniques, considering objectives, types of defects to be identified, and the role of these techniques within the software life cycle.

Terms

dynamic testing, static testing.

This session looks at static testing techniques. These techniques are referred to as "static" because the software is not executed; rather the specifications, documentation and source code that comprise the software are examined in varying degrees of detail.

There are two basic types of static testing. One of these is people-based and the other is tool-based. People-based techniques are generally known as "reviews" but there are a variety of different ways in which reviews can be performed. The tool-based techniques examine source code and are known as "static analysis". Both of these basic types are described in the sections below.

What are Reviews?

"Reviews" is the generic name given to people-based static techniques. More or less any activity that involves one or more people examining software work products (including code) could be called a review. There are a variety of different ways in which reviews are carried out across different organisations and in many cases within a single organisation. Some are very formal, some are very informal, and many lie somewhere between the two. The chances are that you have been involved in reviews of one form or another.

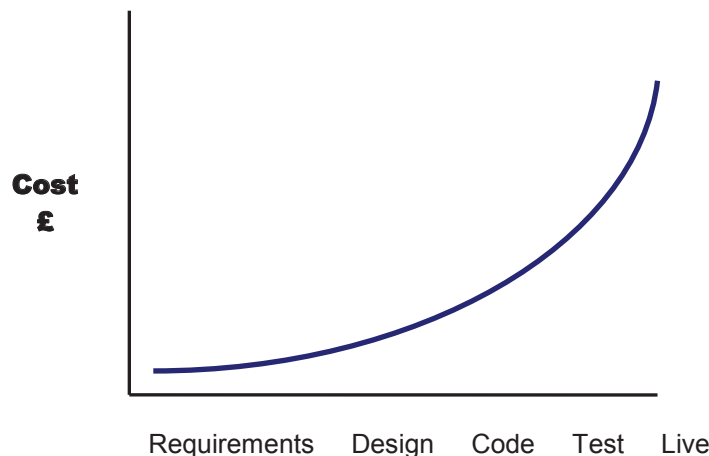
One person can perform a review of his or her own work or of someone else's work. However, it is generally recognised that reviews performed by only one person are not as effective as reviews conducted by a group of people all examining the same document (or whatever it is that is being reviewed).

Benefits of reviews

There are many benefits from reviews in general. They can improve software development productivity and reduce development timescales. They also reduce defects, which in turn leads to a reduction in test execution time and cost. They can lead to lifetime cost reductions throughout the maintenance of a system over its useful life. All this is achieved (where it is achieved) by finding and fixing defects in the products of development phases before they are used in subsequent phases. In other words, reviews find defects in specifications and other documents (including source code) which can then be fixed before those specifications are used in the next phase of development.

Reviews generally reduce defect levels and lead to increased quality. This can also result in improved customer relations. Reviews can also find omissions, for example in requirements, which are unlikely to be found during dynamic testing.

Barry Boehm (pronounced Barry Bame) produced a "cost of finding and fixing defects" curve. This curve shows that the earlier we find and fix the defect – the cheaper it is. This is an excellent case for the use of reviews and Inspections within an organisation:



Cost of finding and fixing defects during the development lifecycle: Barry Boehm

Reviews are cost-effective

There are a number of published figures to substantiate the cost-effectiveness of reviews. Freedman and Weinberg quote a ten times reduction in defects that come into testing with a 50% to 80% reduction in testing cost. Yourdon in his book on Structured Walkthroughs found that defects were reduced by a factor of ten. Gilb and Graham give a number of documented benefits for software Inspection, including 25% reduction in schedules, a 28 times reduction in maintenance cost, and finding 80% of defects in a single pass (with a mature Inspection process) and 95% in multiple passes.

Reviews, static analysis and dynamic analysis all have the same objective – to find defects and all three categories of technique are complementary. Different techniques can however find different types of defect effectively and efficiently.

It is worth making the distinction here that reviews and static analysis find defects whereas dynamic testing finds failures.

Typical defects found during reviews

The typical defects that are easier to find during reviews than in dynamic testing are:

- deviations from standards;
- requirement defects;
- design defects;
- insufficient maintainability and incorrect interface specification.

3.2 Review Process

Learning Objectives

LO-3.2.1	K1	Recall the activities, roles and responsibilities of a typical formal review.
LO-3.2.2	K2	Explain the differences between different types of reviews: informal review, technical review, walkthrough and inspection.
LO-3.2.3	K2	Explain the factors for successful performance of reviews.

Terms

entry criteria, formal review informal review, inspection, metric, moderator, peer review, reviewer, scribe, technical review, walkthrough.

There are different types of review, each with their own distinct characteristics. For example, an informal review is characterised by there being no written instructions for reviewers and no

meeting of reviewers. Another type of review, called inspection, is characterised by there being formal instructions and a documented inspection process for reviewers to follow and possibly two meetings to attend.

Reviews of each type can in themselves vary in formality and this will depend on various aspects such as maturity of the organisation, legal or regulatory needs or specific company standards.

The way the review is carried out is dependent on the review's objectives:

- finding defects;
- gaining understanding;
- discussion;
- coming to a decision by consensus.

3.2.1 Activities of a Formal Review

A typical formal review has the following main activities:

Planning: an important part of the review process where people are selected and associated roles given, entry and exit criteria is set for a more formal type of review such as Inspection. Also the various parts of the document are selected (or all of it) for review.

Kick-off: the document(s) are distributed with a clear understanding of the objectives, process and time scales. Entry criteria are checked for the more formal review.

Individual preparation: people prepare by individually checking the documents before the review meeting. They make notes about defects, questions, comments and process improvements. The preparation or individual checking is usually where the greatest value is gained from a review process. Each person spends time on the review document (and related documents), becoming familiar with it and/or looking for defects.

Review meeting: the meeting can take various forms from a discussion meeting to a more formal logging meeting. Typically a physical meeting of reviewers but a meeting achieved with electronic communication (virtual meetings). Participants may document defects found, make recommendations or make decisions as to the best way forward. Record issues, defects, decisions and recommendations made.

Rework: this part is where the author (typically) makes changes to the original document based on the comments and defects found.

Follow-up: this is the final stage of the process where checks are made as to whether the defects found and logged have been adequately addressed. The more formal review techniques collect metrics on cost (time spent) and benefits achieved and exit criteria are checked. The more formal review techniques include follow-up of the defects or issues found to ensure that action has been taken on everything raised (Inspection does, as do some forms of technical review).

3.2.2 Roles and Responsibilities

A typical formal review will include the following roles:

Managers: have an important role to play in reviews. Even if they are excluded from technical reviews, they can (and should) review management level documents with their peers. They also need to understand the economics of reviews and the value that they bring. They need to ensure that the reviews are done properly, i.e. that adequate time is allowed for reviews in project schedules.

Moderator: this is the person who leads/coordinates the review (therefore sometimes called the leader or coordinator or chairperson). The responsibility of the Moderator is to ensure that the review process works and is responsible for the review taking place. He or she may distribute documents, choose reviewers, mentor the reviewers, call and lead the meeting, perform follow-up and record relevant metrics.

Author: The author of the document being reviewed or Inspected is generally included in the review, although there are some variants that exclude the author. The author actually has the most to gain from the review in terms of learning how to do their work better (if the review is conducted in the right spirit!).

Reviewers: (also known as checkers or inspectors) the reviewers or Inspectors are the people who bring the added value to the process by helping the author to improve his or her document. In some types of review, individual checkers are given specific types of defect to look for to make the process more effective.

Scribe: (also known as a recorder) documents all the issues found, open points and problems found during the meeting.

Looking at software products or related work products from different perspectives and using checklists can make reviews more effective and more efficient. For example, checklists based on different stakeholder perspectives such as users, maintainers, testers, designers and operators, or a checklist of typical requirement or design problems may help uncover previously undetected issues.

3.2.3 Types of Review

We have now established that reviews are an important part of software testing. Testers should be involved in reviewing the development documents that tests are based on, and should also review their own test documentation.

In this section, we will look at different types of reviews, and the activities that are done to a greater or lesser extent in all of them. We will also look at the Inspection process in a bit more detail, as it is the most effective of all review types.

The types of review that we will describe are informal review, walkthrough, technical review and inspection. Note that all but the informal review can be a peer review (but they are not restricted to this). A peer review involves only a peer group (colleagues at the same organisational level).

A document may have more than one type of review associated with it. For example a design specification may begin with a walkthrough to gain understanding of the specification. This then will lead to an informal review by many people and finally an inspection on the most critical parts of the specification.

Informal review

As its name implies, this is very much an ad hoc process. Normally it simply consists of someone giving their document to someone else and asking them to look it over. A document may be distributed to a number of people, and the author of the document would hope to receive back some helpful comments. It is a very cheap form of review because there is no monitoring of metrics, no meeting and no follow-up. It is generally perceived to be useful, and compared to not doing any reviews at all, it is. However, it is probably the least effective form of review (although no one can prove that since no measurements are ever done!). The documentation of the review may take place. The usefulness of this review is variable and depends on the reviewer and importance placed on the review itself.

Walkthrough

A walkthrough is typically led by the author of a document, for the purpose of educating the participants about the content so that everyone understands the same thing, finding defects is also an objective – but could be classed as a secondary objective. A walkthrough may include "dry runs" of business scenarios to show how the system would handle certain specific situations. For technical documents, it is often a peer group technique. It tends to be an open-ended session and may vary in formality.

Technical review

A technical review may have varying degrees of formality. This type of review does focus on technical issues and technical documents. A technical review should exclude senior managers because of the adverse affect they may have on some people (some may keep quiet, some may become too vocal, either way this is undesirable).

The success of this technical reviews typically depends on the individuals involved - they can be very effective and useful, but sometimes they are very wasteful (especially if the meetings are not well disciplined), and can be rather subjective. Often this level of review will have some documentation, even if just a list of issues raised. Sometimes metrics will be kept. This type of review can find important defects, but can also be used to resolve difficult technical problems, for example deciding on the best way to implement a design. Checklists are sometimes used and it is ideally led by a moderator to keep the meeting on track.

A review report may be produced that includes a list of the findings, the verdict of the meeting on whether or not the software product meets its requirements and, if appropriate, recommendations related to the findings.

The main purposes of these vary from finding defects, discussion, making decisions evaluate alternatives and solve technical problems.

Inspection

An Inspection is the most formal of the formal review techniques. There are strict entry and exit criteria to the Inspection process, it is led by a trained leader or moderator (not the author), and there are defined roles for searching for defects based on defined rules and checklists. Metrics are a required part of the process. More detail on Inspection is given later in this session.

In Inspection, it is not just the document under review that is given out in advance, but also source or predecessor documents. The number of pages to focus the Inspection on is closely controlled, so that Inspectors (checkers) check a limited area in depth - a chunk or sample of the whole document. The instructions given to checkers are designed so that each individual checker will find the maximum number of unique defects. Special defect-hunting roles are defined, and Inspectors are trained in how to be most effective at finding defects.

In typical reviews, sometimes the reviewers have time to look through the document before the meeting, and some do not. The meeting is often difficult to arrange and may last for hours.

In Inspection, it is an entry criterion to the meeting that each checker has done the individual checking. The meeting is highly focused and efficient. If it is not economic, then a meeting may not be held at all, and it is limited to two hours.

In a typical review, there is often a lot of discussion, some about technical issues but much about trivia. Comments are often mainly subjective, along the lines of "I don't like the way you did this" or "Why didn't you do it this way?"

In Inspection, the process is objective. The only thing that is permissible to raise as an issue is a potential violation of an agreed Rule (the Rulesets are what the document should conform to). Discussion is severely curtailed in an Inspection meeting or postponed until the end. The Leader's role is very important to keep the meetings on track and focused and to keep pulling people away from trivia and pointless discussion.

Many people keep on doing reviews even if they don't know whether it is worthwhile or not. Every activity in the Inspection process is done only if its economic value is continuously proven.

Below we have included some additional information about Inspection. This goes above and beyond what is in the syllabus so the notes with this shaded background contain additional information that is not examinable.

Inspection is more

Note that this section contains additional information on inspection that is not required by the syllabus and therefore will not be examined in the BCS / ISTQB Certificate examination.

Inspection contains many mechanisms that are additional to those found in other formal reviews. These include the following:

- entry criteria, to ensure that we don't waste time inspecting an unworthy document;
- training for maximum effectiveness and efficiency;
- optimum checking rate to get the greatest value out of the time spent by looking deep;
- prioritising the words: inspect the most important documents and their most important parts;
- standards are used in the inspection process (there are a number of inspection standards also);
- the Leader checks that editing has been completed (follow-up);
- process improvement is built in to the inspection process;
- exit criteria ensure that the document is worthy of being used "downstream" and that the inspection process was carried out correctly.

One of the most powerful exit criteria is the quantified estimate of the remaining defects per page. This may be say 3 per page initially, but can be brought down by orders of magnitude over time.

Inspection is better

Typical reviews are probably only 10% to 20% effective at detecting existing defects. The return on investment is usually not known because no one keeps track even of their cost.

When inspection is still being learned, its effectiveness is around 30% to 40% (this is demonstrated in inspection training courses). Once inspection is well established and mature, this process can find up to 80% of defects in a single pass, 95% in multiple passes. The return on investment ranges from 6 hours to 30 hours for every hour spent. (Source: "Software Inspection", Tom Gilb, Dorothy Graham).

The Inspection process

The Inspection Leader performs the planning activities. A Kickoff meeting is held to "set the scene" about the documents and the process.

The Individual Checking is where most of the benefits are gained. 80% or more of the defects found will be found in this stage.

A meeting is held (if economic). The editing of the document is done by the author or the person now responsible for the document. This involves redoing some of the activities that produced the document initially, and it also may require Change Requests to documents not under the control of the editor. Process improvement suggestions may be raised at any time, for improvements either to the inspection process or to the development process.

The document must pass through the Exit gate before it is allowed to leave the inspection process. There are two aspects to investigate here: is the product document now ready (e.g. has some action been taken on all issues logged), and was the inspection process carried out properly? For example, if the checking rate was too fast, then the checking has not been done properly.

A gleaming new improved document is the result of the process, but there is still a "blob" on it. It is not economic to be 100% effective in inspection. At least with inspection you consciously predict the levels of remaining defects rather than fallaciously assuming that we have found them all!

How the checking rate enables deep checking in Inspection

There is a dramatic difference of Inspection to normal reviews, and that is in the depth of checking. This is illustrated by the picture of a document. Initially there are no defects visible. Typically in reviews, the time and size of document determine the checking rate. So for example if you have 2 hours available for a review and the document is 100 pages long, then the checking rate will be 50 pages per hour. (Any two of these three factors determine the third.)

This is equivalent to "skimming the surface" of the document. We will find some defects - in this example we have found one major and two minor defects. Our typical reaction is now to think: "This review was worthwhile wasn't it - it found a major defect. Now we can fix that and the two other minor defects, and the document will now be OK." Think: are we missing anything here?

Inspection is different. We do not take any more time, but it is the optimum rate for the type of document that is used to determine the size of the document that will be checked in detail. So if the optimum rate is one page per hour and we have two hours, then the size of the sample or chunk will be 2 pages.

(Note that the optimum rate needs to be established over time for different types of document and will depend on a number of factors, and it is based on prioritised words (logical page rather than physical page). Of course it doesn't take an hour just to read a single page, but the checking done in Inspection includes comparing each paragraph or sentence on the target page with all source documents, checking each paragraph or phrase against relevant rule sets, both generic and specific, working through checklists for different role assignments, as well as the time to read around the target page to set the context. If checking is done to this level of thoroughness, it is not at all difficult to spend an hour on one page!)

How does this depth-oriented approach affect the defects found? On the picture, we have gone deep in the Inspection on a limited number of pages. We have found the major one found in the other review plus two (other) minors, but we have also found a deep-seated major defect, which we would never have seen or even suspected if we had not spent the time to go deep. There is no guarantee that the most dangerous defects are lying near the surface!

When the author comes to fix this deep-seated defect, he or she can look through the rest of the document for similar defects, and all of them can then be corrected. So in this example we will have corrected 5 major defects instead of one. This gives tremendous leverage to the Inspection process - you can fix defects you didn't find!

Inspection surprises

To summarise the Inspection process, there are a number of things about Inspection which surprise people. The fundamental importance of the Rules is what makes Inspection objective rather than a subjective review. The Rules are democratically agreed as applying (this helping to defuse author defensiveness) and by definition a defect is a Rule violation.

The slow checking rates are surprising, but the value to be gained by depth gives far greater long-term gains than surface-skimming review that miss major deep-seated problems.

The strict entry and exit criteria help to ensure that Inspection gives value for money.

The logging rates are much faster than in typical reviews (30 to 60 seconds; typical reviews log one thing in 3 to 10 minutes). This ensures that the meeting is very efficient. One reason this works is that the final responsibility for all changes is fully given to the author, who has total responsibility for final classification of defects as well as the content of all fixes.

More information on Inspection can be found in the book *Software Inspection*, Tom Gilb and Dorothy Graham, Addison-Wesley, 1993, ISBN 0-201-63181-4.

3.2.4 Success Factors for Reviews

Some of the most common success factors for reviews are listed below.

- The review objectives must be clearly stated
- The right people need to be involved
- Testers are valued reviewers who contribute to the review and also learn about the product, thereby enabling them to prepare tests earlier.
- Defects found are “welcome” and expressed objectively.
- People issues must be dealt with - the right attitude must be evident from both the author and the reviewer.
- The review is conducted in an atmosphere of trust (that the outcome will not be used for the evaluation of the participants).
- Review techniques are applied that are suitable to achieve the objectives and to the type and level of work product and the reviewers.
- Checklists and roles are used to increase efficiency and effectiveness.
- Training is given in the various review techniques, especially the more formal techniques such as Inspections.
- Management supports the review process – this is achieved by giving adequate time for the review to take place on the project and the appropriate people are made available.
- Process improvement and learning are key factors in the review process.

Reasons why reviews don't always work?

Reviews are not always successful. They are sometimes not very effective, so defects that could have been found slip through the net. They are sometimes very inefficient, so that people feel that they are wasting their time. Often insufficient thought has gone into the definition of the review process itself - it just evolves over time.

One of the most common causes for poor quality in the review process is lack of training, and this is more critical the more formal the review.

Another problem with reviews is dealing with documents that are of poor quality. Entry criteria to the review or Inspection process can ensure that reviewers' time is not wasted on documents that are not worthy of the review effort.

A lack of management support is a frequent problem. If managers say that they want reviews to take place but don't allow any time in the schedules for the, this is only "lip service" not commitment to quality.

Long-term, it can be disheartening to become expert at detecting defects if the same defects keep on being injected into all newly written documents. Process improvements are the key to long-term effectiveness and efficiency.

3.3 Static Analysis

Learning Objectives

- | | | |
|----------|----|-----------------------------------------------------------------------------------------------------------------|
| LO-3.3.1 | K1 | Recall typical defects and errors identified by static analysis and compare them to review and dynamic testing. |
| LO-3.3.2 | K2 | Describe, using examples, the typical benefits of static analysis. |
| LO-3.3.3 | K1 | List typical code and design defects that may be identified by static analysis tools. |
-

Terms

compiler, complexity, control flow, data flow, static analysis.

What can static analysis do?

Static analysis is a form of automated testing. It can check for violations of standards and can find things that may or may not be defects. Static analysis is descended from compiler technology. In fact, many compilers may have static analysis facilities available for developers to use if they wish. There are also a number of stand-alone static analysis tools for various different computer programming languages. Like a compiler, the static analysis tool analyses the code without executing it, and can alert the developer to various things such as unreachable code, undeclared variables, etc.

Static analysis tools can also compute various metrics about code such as cyclomatic complexity.

As with reviews, static analysis finds defects rather than failures. The concept of static analysis tools is that they analyse program code.

The value of static analysis is:

- finding defects prior to test execution;
- early warning about suspicious aspects of the code;
- finding defects that would be hard to find using dynamic techniques;
- detecting inconsistencies in software models;
- improved maintainability of the code and design;
- prevention of defects when lessons learned are implemented.

Typical defects discovered by static analysis tools include:

- referencing a variable with an undefined value;
- inconsistent interfaces between modules and components;
- variables that are never used;
- unreachable (dead) code;
- missing and erroneous logic (potentially infinite loops/array bound violations);
- overly complicated constructs;
- programming standards violations;
- security vulnerabilities;
- syntax violations of code and software models.

Static analysis tools are typically used by developers (checking against predefined rules or programming standards) before and during component and integration testing or when checking-in code to configuration management tools. They may also be used by designers during software modelling. Static analysis tools may produce a large number of warning messages, which need to be well-managed to allow the most effective use of the tool.

Compilers may offer some support for static analysis, including the calculation of metrics.

There are two types of static analysis:

- data flow analysis
- control flow analysis

These are explained below.

Data flow analysis

Data flow analysis is the study of program variables. A variable is basically a location in the computer's memory that has a name so that the programmer can refer to it more conveniently in the source code. When a value is put into this location, we say that the variable is "defined". When that value is accessed, we say that it is "used".

For example, in the statement " $x = y + z$ ", the variables y and z are used because the values that they contain are being accessed and added together. The result of this addition is then put into the memory location called " x ", so x is defined.

The significance of this is that static analysis tools can perform a number of simple checks. One of these checks is to ensure that every variable is defined before it is used. If a variable

is not defined before it is used, the value that it contains may be different every time the program is executed and in any case is unlikely to contain the correct value. This is an example of a data flow error (it should be called a “defect” but the tools refer to them as “data flow errors”). Another check that a static analysis tool can make is to ensure that every time a variable is defined it is used somewhere later on in the program. If it isn’t, then why was it defined in the first place? This is known as a data flow anomaly and although it can be a perfectly harmless defect, it can also indicate that something more serious is wrong.

Control flow analysis

Control flow analysis can find infinite loops, inaccessible code, and many other suspicious aspects. However, not all of the things found are necessarily defects; defensive programming may result in code that is technically unreachable.

Limitations and advantages

Static analysis has its limitations. It cannot distinguish “fail-safe” code from real defects or anomalies, and may create a lot of spurious failure messages. Static analysis tools do not execute the code, so they are not a substitute for dynamic testing, and they are not related to real operating conditions.

However, static analysis tools can find defects that are difficult to see and they give objective quality information about the code. We feel that all developers should use static analysis tools, since the information they can give can find defects very early when they are very cheap to fix.

Session 4

Test Design Techniques

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

2

The Process of Testing

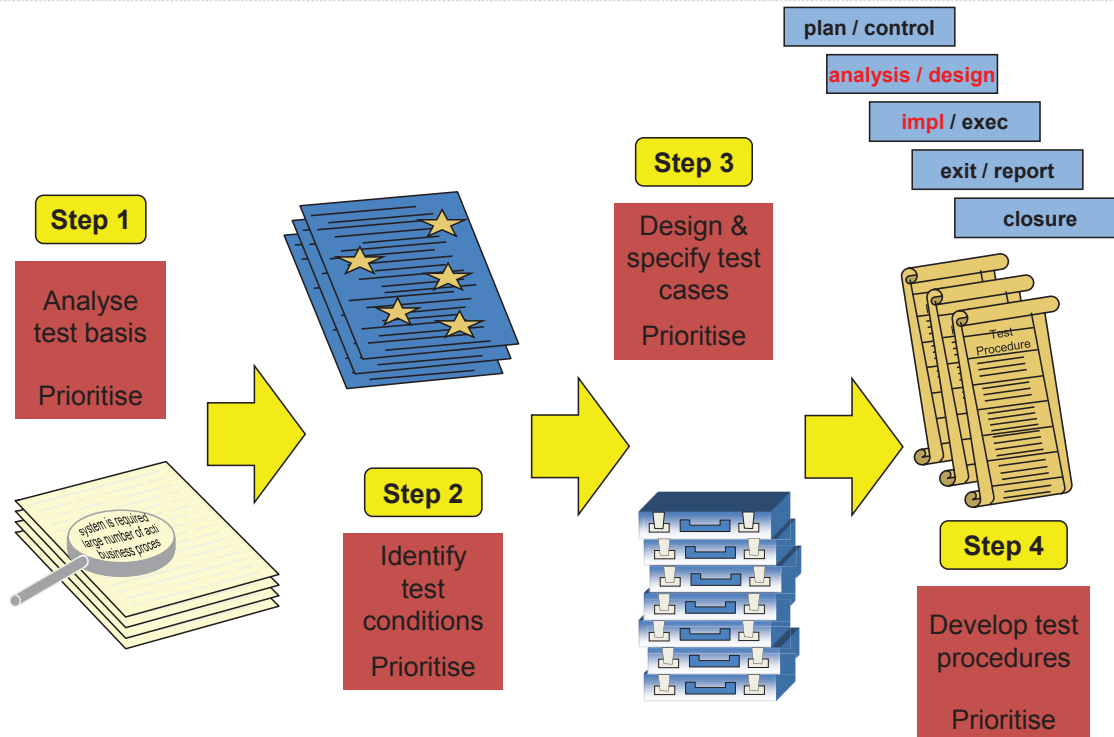
- may be performed...



- level of formality depends on
 - context of testing, such as
 - ▶ maturity of development and test processes
 - ▶ time constraints
 - ▶ people involved
 - ▶ safety or regulatory requirements

3

4 Steps of the Test Development Process



4

Step 1: Analyse Test Basis

- study the test basis
 - e.g. requirement, functional, or design specifications, code, user manuals, business process descriptions
- evaluate testability, assess risk and prioritise
 - can the requirement/function/etc. be tested?
 - which requirements, functions, etc. need more testing
 - select appropriate techniques



Example: online orders can be made at anytime, but will be processed on the next **working day**. Payment must be made using PayPal or **standard** credit/debit cards. Orders will be dispatched **within** 2 days. Customers will be notified via **email** if a **partial** order has been sent and can return their order **within** 28 days and obtain a **full** refund.

5

Step 2: Identify Test Conditions

- list the test conditions that we can identify and prioritise
 - use the identified techniques or brainstorm
 - there may be many conditions for each requirement, function or process.
 - may be a mixture of general (high level) conditions and more detailed (low level) conditions
 - must ensure most important conditions are covered



Example test conditions:

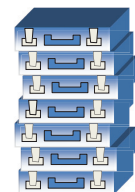
1. place an order (all items in stock) on a weekday (high level)
2. place an order on a public holiday (high level)
3. pay by PayPal (low level)
4. return a partial order and obtain full refund (high level)...

6

Step 3: Design & Specify Test Cases

- select the test condition(s) to be tested
 - important ones first
- specify test cases
 - input values
 - execution pre-conditions
 - expected results
 - execution post-conditions

} prioritise



TC name: place an order - testing book

Precondition: Software Test Automation book in stock

Input: order "Software Test Automation book"

Result: correct book included in order

Post-condition: stock level reduced by 1

TC name: place order on bank holiday

Precondition: calendar defines holidays

Input: Place order on "Saturday"

Result: order processed on Tuesday

Post-condition: system stores information until Tuesday

7

Test Design Tools

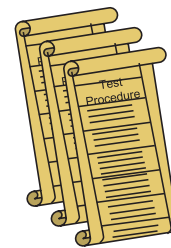
- generate test inputs
 - from a formal specification (test basis)
 - from a CASE repository (design model)
 - from code (e.g. code not covered yet)
- generate “limited” expected outcomes
 - from an Oracle (e.g. existing system)
- benefits
 - save time, increased thoroughness (but can generate too many tests!)



8

Step 4: Develop Test Procedures

- combine test cases into procedures
 - a specific sequence of test cases
 - combining all the actions, inputs, preconditions, expected results and post-conditions
- prioritise and determine order of execution
 - test procedure specification generated
 - ▶ used manually or automated for regression testing (script)



Pre-condition: 9 STA books in stock
Pre-condition: existing customer Jones
Input: Jones orders 10 STA books
Input: order day = Monday
Input: payment method = PayPal
Result: partial order of 9 STA books sent (cont..)

(continued)
Result: order sent on Tuesday
Result: Jones notified via email
Result: PayPal debited
... ..
Post-condition: STA books available = 0
Post-condition: Order more STA books

9

Test Data Preparation Tools

- data manipulation
 - selected from existing databases or files
 - created according to some rules
 - edited from other sources
 - safeguards confidentiality when using live data
 - ▶ e.g. “data protection act” requires anonymity of personal data
- test data is used during test execution



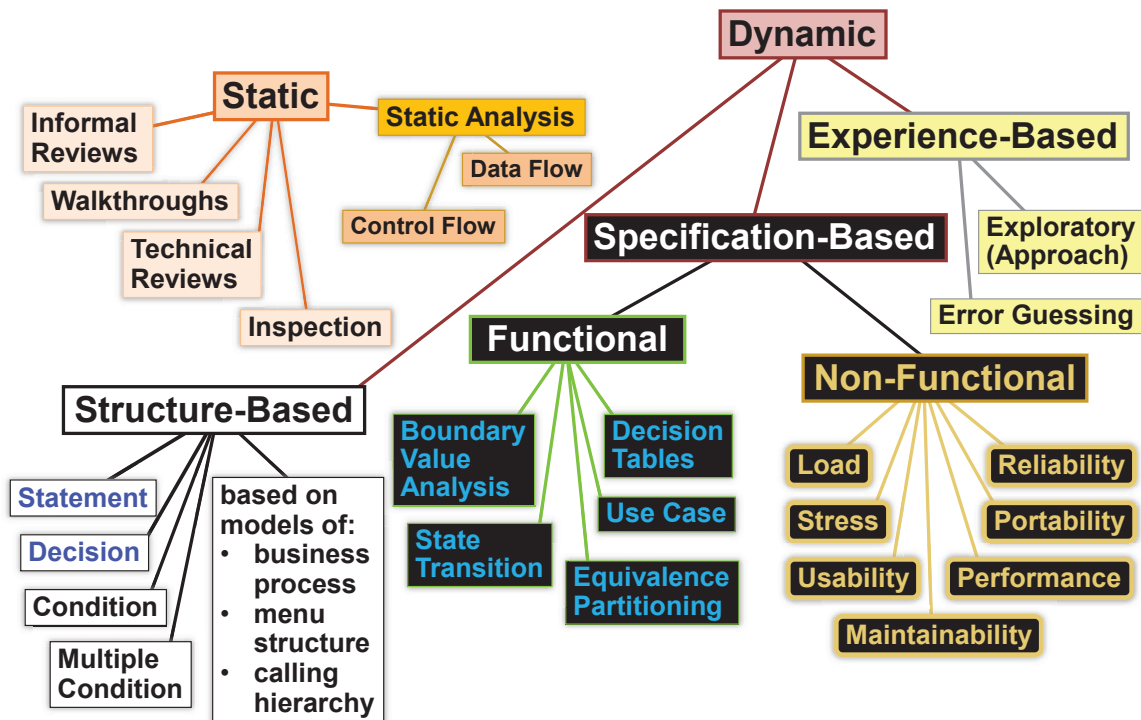
10

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

11

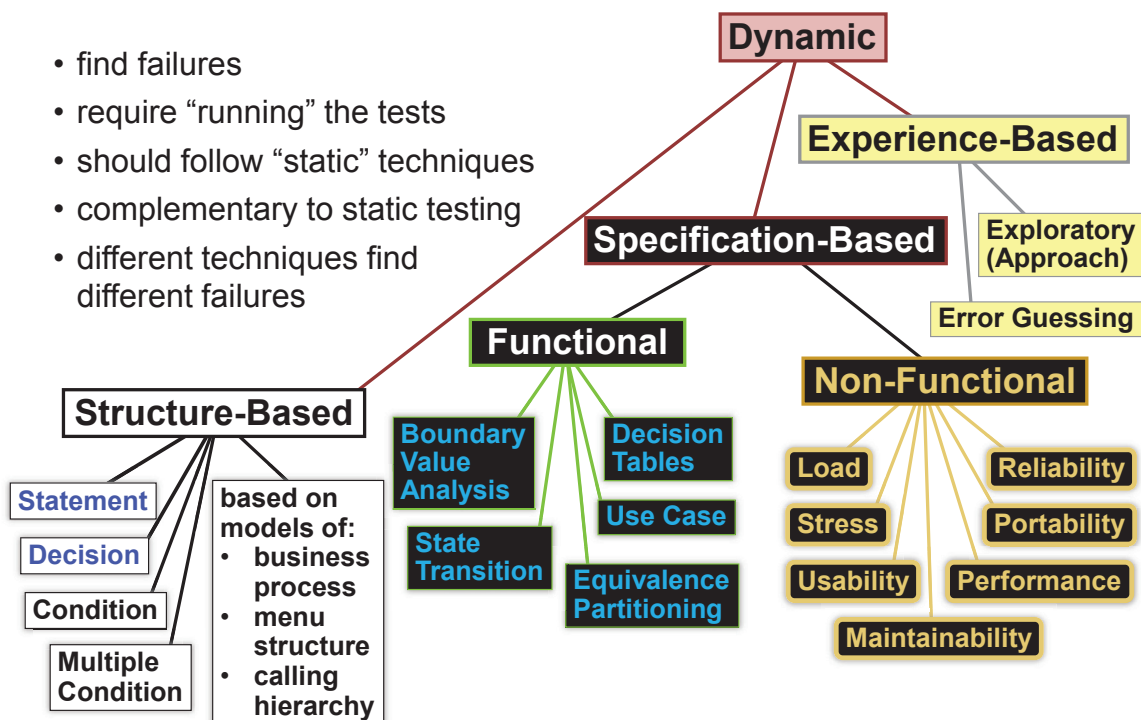
Static and Dynamic Testing



12

Static and Dynamic Testing

- find failures
- require “running” the tests
- should follow “static” techniques
- complementary to static testing
- different techniques find different failures



13

What is a Test Design Technique?

- a process to help identify test conditions and test cases
- thought processes that help us select test cases more intelligently
 - to successfully find failures (leading to defect discovery)
- addressing principle number 2

2. Exhaustive testing is impossible (in practice)

- by selecting a subset of all possible tests

Testing should be thorough and systematic

14

Benefits of Test Design Techniques

- different people: same probability find defects
 - gain some independence of thought
- effective testing: find more defects
 - focus attention on specific types of fault
- efficient testing: find defects with less effort
 - avoid duplication and can be quicker
- 3 types of technique
 - ▶ specification-based
 - ▶ structure-based
 - ▶ experience-based

Techniques make testing more effective and more efficient

15

Specification-based Techniques (Black Box)

- use models (descriptions) of what the software should do
 - a specification (formal model)
 - our understanding of what the software should do (informal model)
- analysis of the test basis documentation (functional or non-functional) without reference to structure
- test cases derived systematically



16

Structure-based Techniques (White Box)

- uses a model (description) of the software
 - based on how the software is constructed
 - typically the source code, design or menu structure
- analysis of internal structure of component or system
- coverage of the construct by existing test cases can be measured
- further test cases derived systematically to increase coverage



17

Experience-based Techniques (Black Box)

- knowledge is used to derive test conditions and test cases
 - knowledge of the testers, developers, users, and other stakeholders
 - knowledge about the software, its use and its environment
 - knowledge about likely defects and their distribution
 - heuristics (rules of thumb)



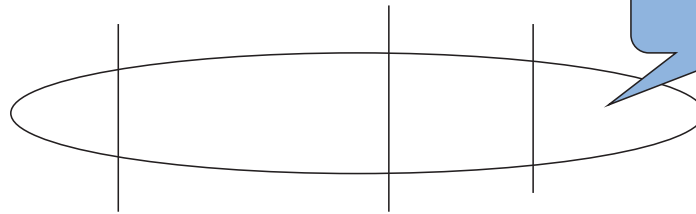
18

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
 - 4.3.1 Equivalence Partitioning & Boundary Value Analysis
 - 4.3.2 Decision Table Testing
 - 4.3.3 State Transition Testing
 - 4.3.4 Use Case Testing
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

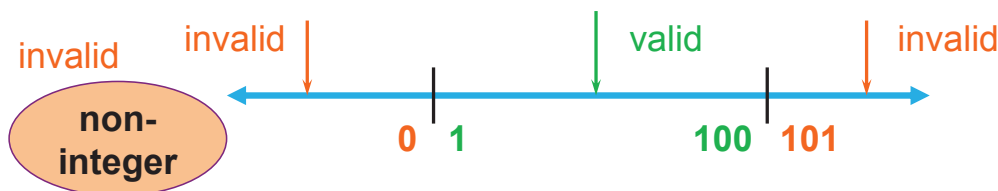
19

Equivalence Partitioning (EP)



also known as an
"equivalence
class"

- divide (partition) the inputs and outputs into areas that are the same (equivalent)
- assumption: if one value works, all will work
- one from each partition better than all from one



20

Equivalence Partitioning (EP) Example

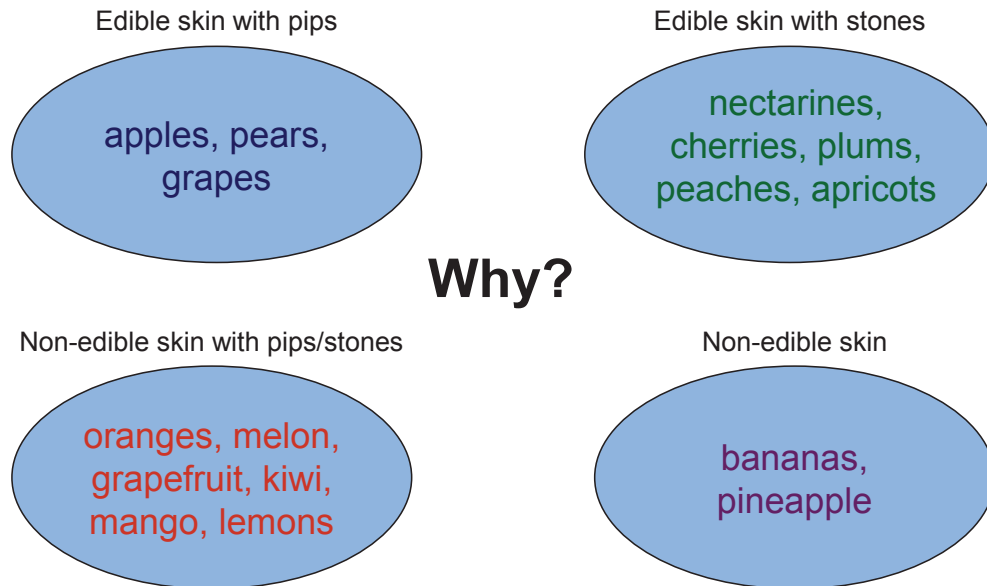
a supermarket has a variety of fruit on its shelves

- | | |
|--------------|-------------|
| • apples | • cherries |
| • oranges | • kiwi |
| • pears | • mango |
| • bananas | • plums |
| • nectarines | • peaches |
| • grapes | • apricots |
| • melon | • pineapple |
| • grapefruit | • lemons |

put these into
partitions of your
choice

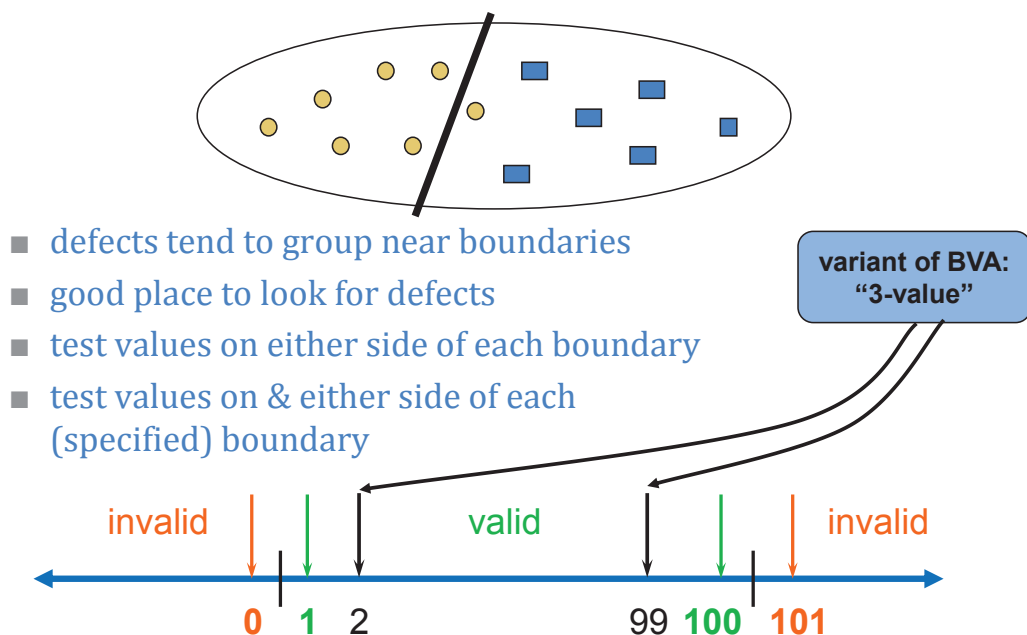
21

Equivalence Partitioning (EP) Solution



22

Boundary Value Analysis (BVA)



23

Example Specification (Part 1)

A new hard-drive recording system is required for households with up to 6 people. Each new household member will be set up with a unique name (4 – 8 characters), a pin number (4 digits) and an age category is chosen from a list (0-11, 12-17, 18+)

apply Equivalence Partitioning and Boundary Value Analysis...

member set up

name :

pin :

age :

24

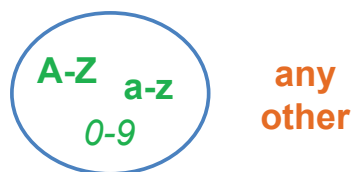
Member Name and Pin Number

Member name

Number of characters:

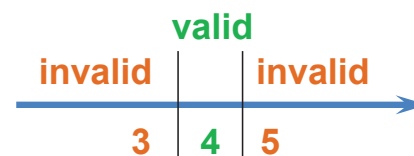


Valid characters:



Pin number

Number of digits:



Valid digits:



25

Age Category and Number of Members

Age category:



there are no boundaries because it is a list box

Number of members allowed:



26

Design Test Cases

Test Case	Description	Expected Outcome
1	Name: John1 Pin no: 1234 Age: 18+	adult member accepted
2	Name: Archie99 Pin no: 9999 Age: 0-11	child member accepted
3	Name: JohnSmith Pin no: 9998 Age: 12-17	error: name too long
	...	

27

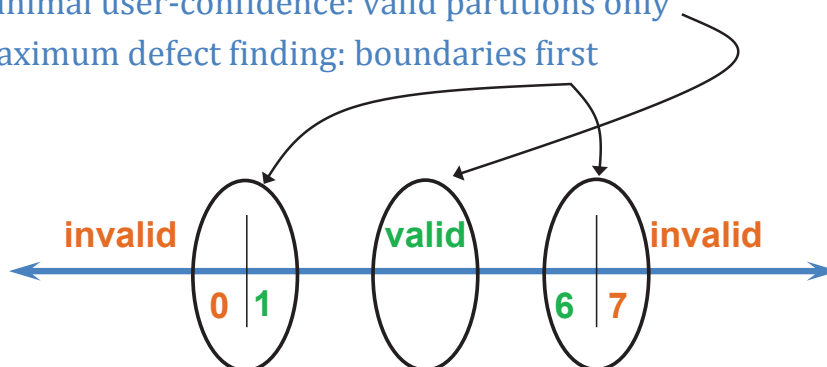
Why Do Both EP and BVA?

- if you do boundaries only, you have covered all the partitions as well
 - technically correct and may be OK if everything works correctly!
 - if the test fails, is the whole partition wrong, or is a boundary in the wrong place – you will have to test inside the middle of the partition anyway
 - testing only extremes may not give confidence for typical use scenarios (especially for users)
 - boundaries may be harder (more costly or difficult) to set up

28

Test Objectives?

- for a thorough approach:
 - all partitions and boundaries (valid and invalid)
- under time pressure, depends on your test objective
 - minimal user-confidence: valid partitions only
 - maximum defect finding: boundaries first



29

Question: EP

Car insurance premiums are based on age of driver. 17 to 25 years inclusive are charged rate A, 26 to 50 are rate B, and >50 rate C



Which of the following are in different VALID equivalence classes?

a) 19, 23, 44

b) 32, 47, 49

c) 17, 25, 50

d) 22, 33, 55

30

Question: 2-value BVA

Charges for downloading files is based on the time it takes. Files downloading in less than 2 minutes are charged €1; files taking from 2:00 to 5:00 minutes inclusive: charged €2, longer downloads: charged €3.



Using 2-value Boundary Value Analysis, what length of download times should be tested?

a) 2:00, 2:01, 5:00, 5:01

b) 1:59, 2:00, 5:00, 5:01

c) 1:59, 2:00, 4:59, 5:00

d) 2:00, 2:01, 4:59, 5:00

31

Question: 3-value BVA

A holiday club restricts membership to those aged between 18 and 30 inclusive. Using 3-value Boundary Value Analysis, which boundary values should be identified?



- a) 18, 19, 20, 28, 29, 30
- b) 16, 17, 18, 30, 31, 32
- c) 17, 18, 19, 29, 30, 31
- d) 18, 19, 21, 29, 30, 35

32

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
 - 4.3.1 Equivalence Partitioning & Boundary Value Analysis
 - 4.3.2 Decision Table Testing
 - 4.3.3 State Transition Testing
 - 4.3.4 Use Case Testing
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

33

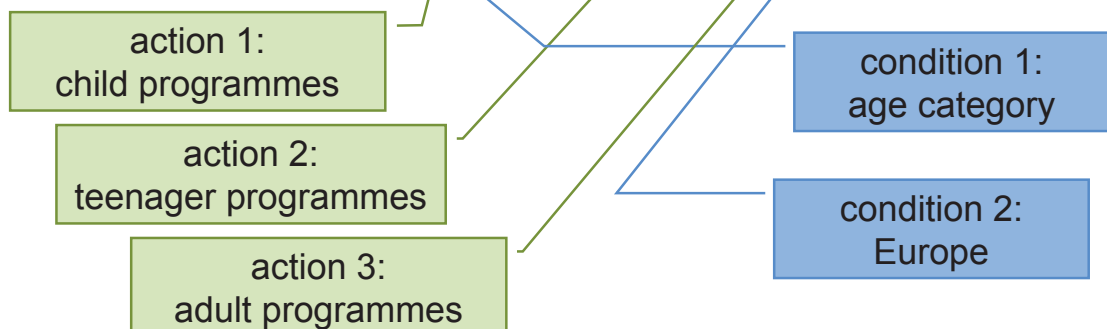
Decision Tables

- used to help explore combinations
 - combinations of inputs, situations or events
 - can be used to specify complex business rules
 - ▶ based on a set of conditions (that can be true or not)
 - helps to ensure that important combinations are not overlooked
- helps testers ensure all combinations / business rules are considered
 - often impractical to test all combinations
 - decision tables help identify a good subset

34

Example Specification (Part 2)

The age category of the member restricts the viewing for that particular member. 0-11 years can only watch “child” programmes, 12-17 years can watch “child” and “teenager” programmes only and 18+ years can watch all programmes (including “adult”). These restrictions are limited to systems being used in Europe.



what are the conditions (causes) and actions?

35

List Conditions and Actions

- list the 'conditions' in the first column of the table
- list the 'actions' under the conditions

conditions	
age category	
Europe	
actions	
child programmes	
teen programmes	
adult programmes	

36

Construct the Decision Table

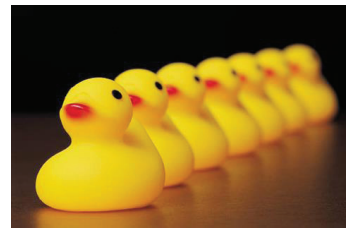
conditions						
age category	0-11	0-11	12-17	12-17	18+	18+
Europe	F	T	F	T	F	T
actions						
child programme	T	T	T	T	T	T
teen programme	T	F	T	T	T	T
adult programme	T	F	T	F	T	T

37

Rationalising a Decision Table

- analysing the decision table may indicate the table can be “rationalised”
 - a condition may result in a particular action, no matter what the other conditions are or
 - if the same set of actions occurs in two columns, a condition may have no effect
- eliminating these ineffective combinations is known as rationalising the decision table

} different ways to consider rationalising



38

Rationalise the Decision Table Example

can this decision table be rationalised?

conditions						
age category	0-11	0-11	12-17	12-17	18+	18+
Europe	F	T	F	T	F	T
actions						
child programme	T	T	T	T	T	T
teen programme	T	F	T	T	T	T
adult programme	T	F	T	F	T	T

39

A Rationalised Decision Table

conditions				
age category	-	0-11	12-17	18+
Europe	F	T	T	T
actions				
child programme	T	T	T	T
teen programme	T	F	T	T
adult programme	T	F	F	T
tags	A	B	C	D

Rationalising is based on assumptions – which may be wrong!

each column is a test condition

40

Design Test Cases

- usually one test case for each column
 - but can be none or several

test	description	expected outcome	tag
1	age 0-11 outside Europe	member can watch any programme	A
2	age 0-11 in Europe	member can only watch child programmes	B
3	age 12-17 in Europe	member can watch child and teen progs	C
4	age 18+ in Europe	member can watch any programme	D

41

Question: Decision Table*



	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Conditions					
delivery	< 3 days	< 3 days	4-7 days	>7-14 days	>7-14 days
weight	light	heavy	don't care	light	heavy
value	< 20	> 50	<50	>50	don't care
Actions					
send by	post	courier	post	post	own truck
insured?	no	yes	no	yes	no

Test case A: a heavy package, worth 65, due in 2 weeks

Test case B: a light package, worth 15, due tomorrow

a) A: post, insured, B: post, insured

b) A: courier, insured; B: post, not insured

c) A: own truck, not insured; B: post, not insured

d) A: own truck, not insured; B: courier, insured

* some rules
omitted for
example
simplicity

42

Contents

4.1 The Test Development Process

4.2 Categories of Test Design Techniques

4.3 Specification-Based or Black-Box Techniques

4.3.1 Equivalence Partitioning & Boundary Value Analysis

4.3.2 Decision Table Testing

4.3.3 State Transition Testing

4.3.4 Use Case Testing

4.4 Structure-Based or White-Box Techniques

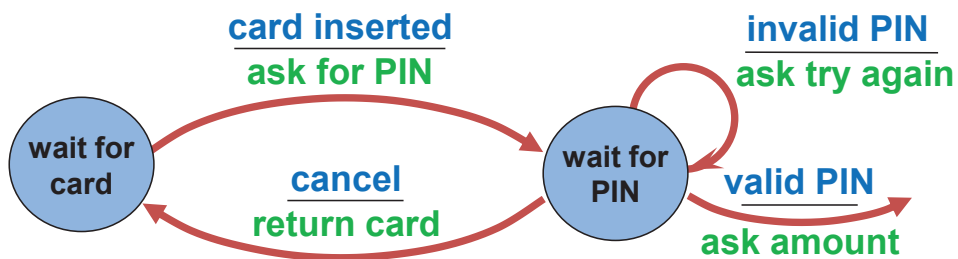
4.5 Experience-Based Techniques

4.6 Choosing Test Techniques

43

State Transition Testing (Analysis)

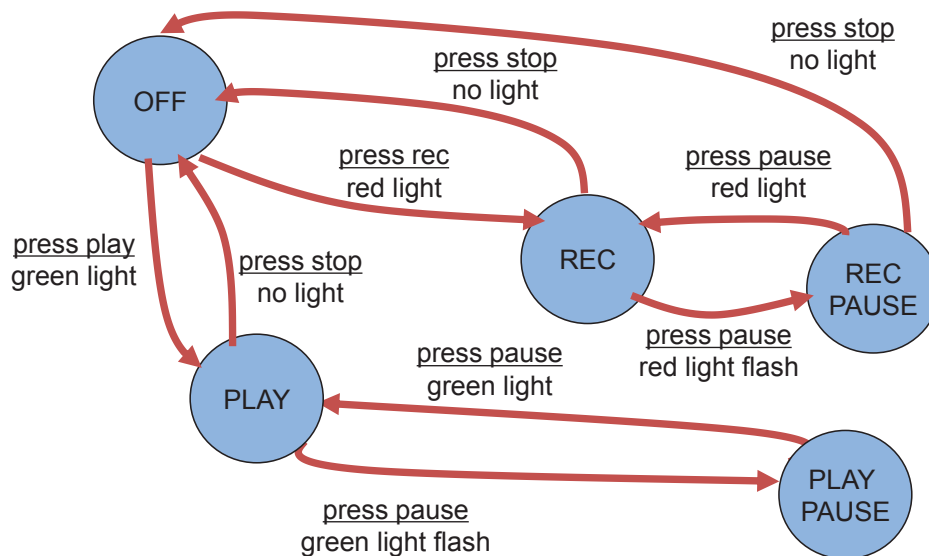
- uses a model that shows:
 - **states** the software may occupy
 - **transitions** between the states
 - **events** that cause the transitions
 - **actions** that result from the transitions



44

Example Specification (Part 3)

When a member is logged onto the system they can watch or record the allowable programmes by using the buttons on the remote. The sequence of actions is described in the diagram below.



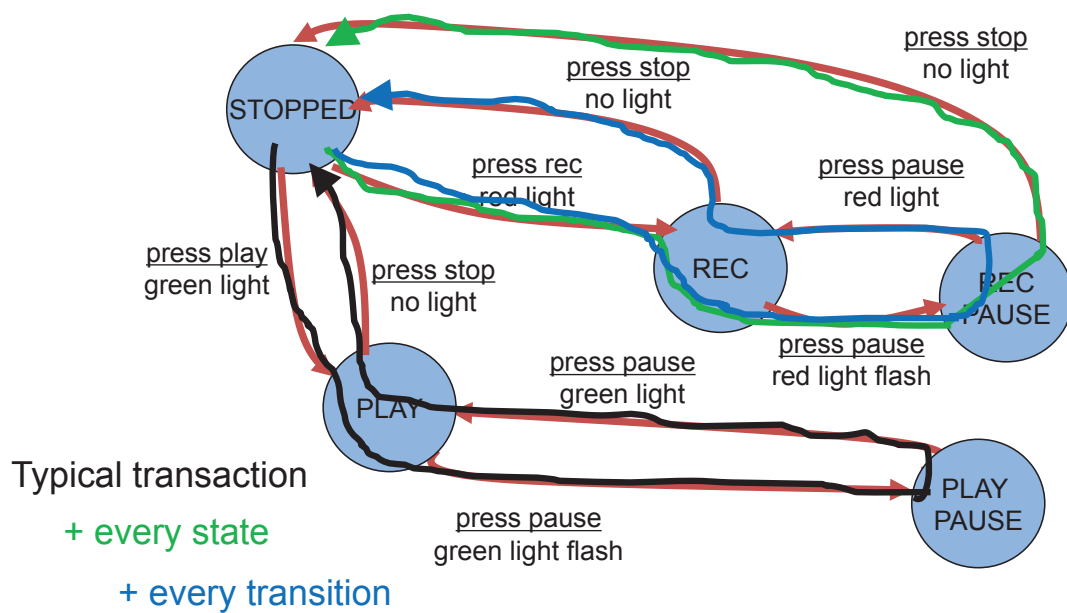
45

State Transition Testing (Design)

- test cases are designed to cover:
 - typical sequence of states
 - every state
 - every transition
 - sequences of transitions
 - ▶ i.e. transition pairs, transition triples, ...
- also design tests for invalid transitions
 - use the state table to identify invalid (or unspecified) transitions

46

State Transition Example



47

State Table Example

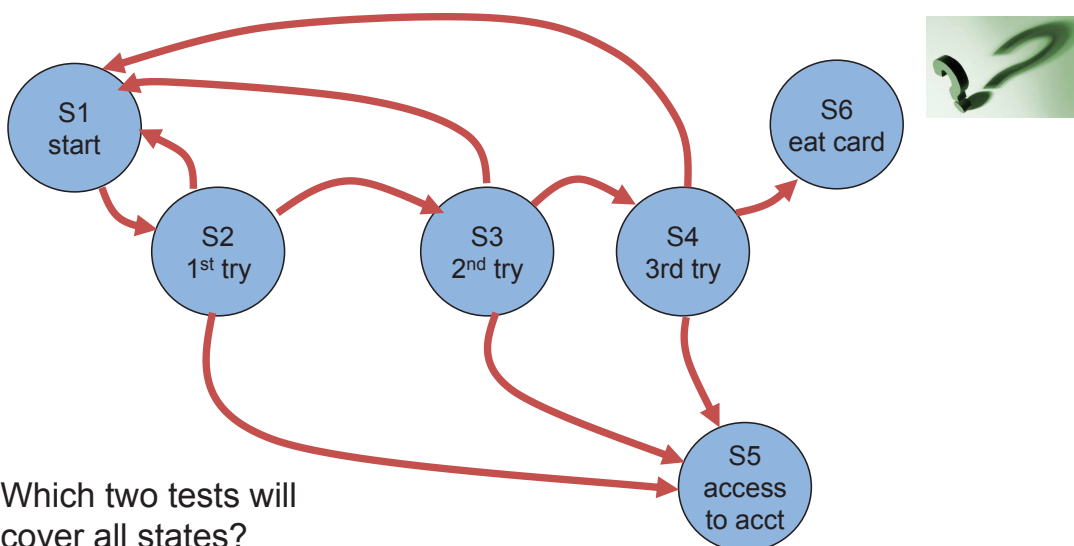
Example invalid test: try to press "rec" when in the play state

states \ events	Press "stop"	Press "play"	Press "rec"	Press "pause"
STOPPED	-	PLAY	REC	-
PLAY	STOPPED	-	-	PLAY PAUSE
RECORD	STOPPED	-	-	REC PAUSE
PLAY PAUSE	-	-	-	PLAY
REC PAUSE	STOPPED	-	-	REC

"-" invalid transitions / spec doesn't say

48

Question: State Transition Graph



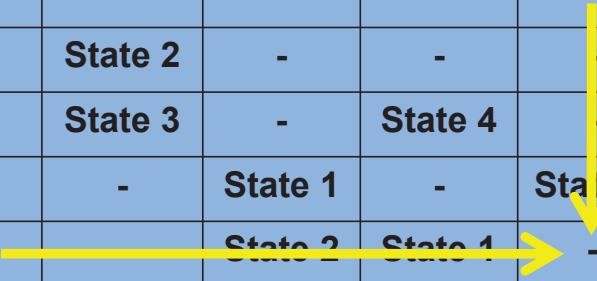

Which two tests will cover all states?

- a) S1 – S2 – S3 – S4 – S5 and S1 – S4 – S6
- b) S1 – S2 – S4 – S5 – S6 and S1 – S2 – S3 – S4 – S5
- c) S1 – S2 – S3 – S4 – S5 and S1 – S2 – S3 – S4 – S6
- d) S1 – S2 – S3 – S4 – S5 and S4 – S6

49

Question: State Table

	Event A	Event B	Event C	Event D
State 1	State 2	-	-	-
State 2	State 3	-	State 4	-
State 3	-	State 1	-	State 4
State 4	-	State 2	State 1	-



Which of the following is an invalid transition?

- a) State 2 event A
- b) State 4 event B
- c) State 2 event C
- d) State 4 event D

50

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
 - 4.3.1 Equivalence Partitioning & Boundary Value Analysis
 - 4.3.2 Decision Table Testing
 - 4.3.3 State Transition Testing
 - 4.3.4 Use Case Testing
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

51

Use Cases

- a description of how a system will be used
 - specified at an abstract level (from business user's perspective)
 - or specified at system level (considering system functions)
- uses "actor(s)" and their interfaces to the system
 - a person (user, operator) or another system
 - if multiple actors, write from perspective of principle actor
- produces some useful result to a system user
- useful for
 - basis for system and acceptance tests
 - involving customers / users
 - finding integration defects through interactions

52

Key Content of a Use Case

- main scenario and any alternative scenarios
 - handling special cases or exceptional conditions
- role name(s) or description of actor(s)
- preconditions
 - must be met before the use case starts
- post-conditions
 - state of system after successful completion
 - observable results (e.g. printouts)

53

Example Use Case

- **Identifier:** UC023
- **Goal:** Download bank account statements
- **Actor:** account owner
- **Preconditions:** account exists with recent transactions not previously downloaded
- **Post-conditions:** file downloaded contains correct statement of transactions.
- **Scenario:**
 - 1 Actor enters user id and password, System displays last login time
 - 2 Actor confirms last login time, System displays current balance of account and menu
 - 3 Actor selects 'Download', System displays download menu
 - 4 Actor ...
- **Alternate:**
 - 1 Actor enters user id and incorrect password System displays error message and first retry request ...

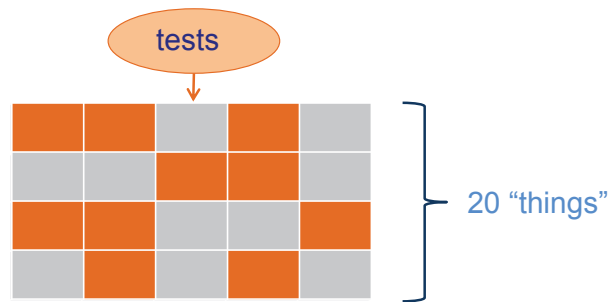
54

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
 - 4.4.1 Statement Testing and Coverage
 - 4.4.2 Decision Testing and Coverage
 - 4.4.3 Other Structure-Based Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

55

The Concept of Coverage



Our tests have covered 10/20 "things" = 50% "thing" coverage where "things" can be:

- lines of code = statements
 - decision outcomes = decisions
 - branches
 - modules
 - menu options
 - functions
- component level
- integration level
- system level

56

Statement Testing

- structural testing technique
 - dynamic technique based on code
 - purpose is to identify test cases that exercise executable statements in the code that have not already been exercised by existing test cases
 - ▶ an executable statement does not include variable declarations and comments.
- example defects
 - wrong message output
 - wrong data used
 - wrong calculation performed

57

Statement Coverage

Statement coverage is normally measured by a software tool.

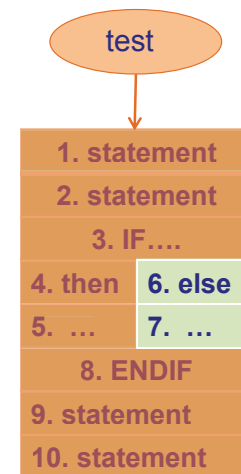
- a measurement technique
- percentage of executable statements exercised by a test suite

$$= \frac{\text{number of statements exercised}}{\text{total number of statements}}$$

- example:

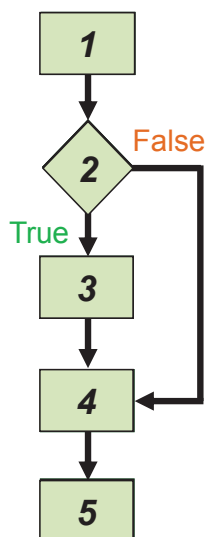
- program has 10 statements
- tests exercise 8 statements

statement coverage = $8/10 = 80\%$



58

Example of Statement Coverage



1	read(n)
2	IF n > 6 THEN
3	n = n * 2
4	ENDIF
5	print n

Statement numbers

Test Case	Input	Expected Output
A	7	14

Test Case	Path Taken	Statement Coverage
A	1, 2, 3, 4, 5	100%

59

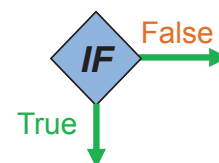
Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
 - 4.4.1 Statement Testing and Coverage
 - 4.4.2 Decision Testing and Coverage
 - 4.4.3 Other Structure-Based Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

60

Decision Testing

- structural testing technique
 - dynamic technique based on code
 - purpose is to identify test cases that exercise decision outcomes in the code that have not already been exercised by existing test cases
 - ▶ decision outcome = “true” and “false” from a decision statement (IF, While, For, Repeat)
- example defects
 - variable not initialised correctly
 - invalid input accepted

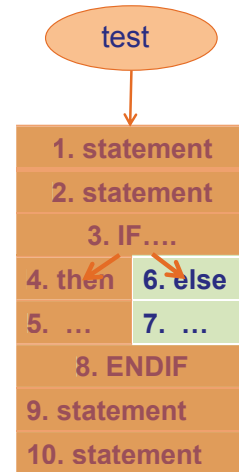


61

Decision Coverage

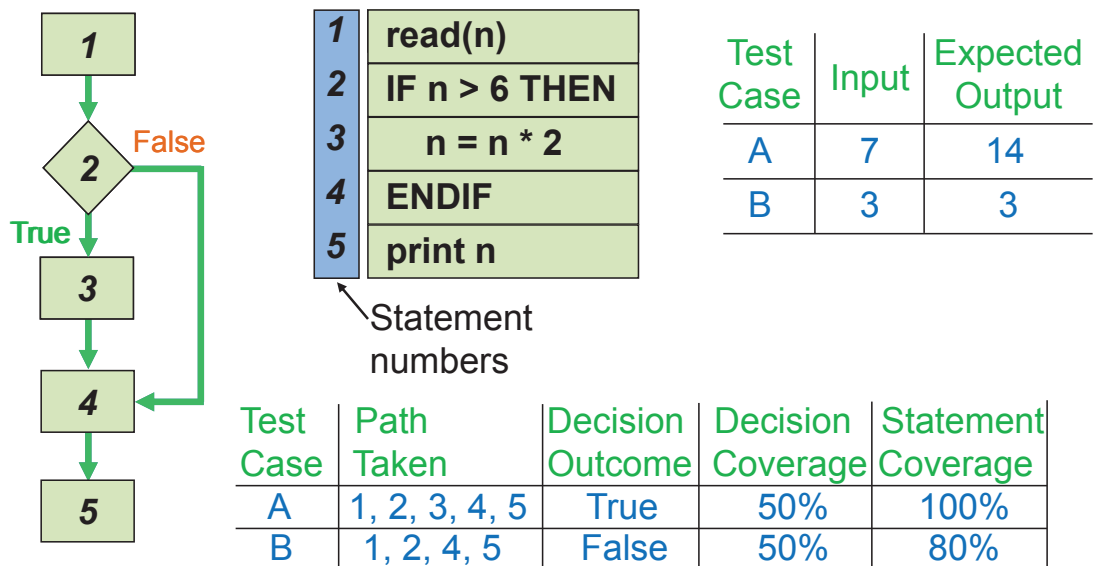
Decision coverage is normally measured by a software tool.

- a measurement technique
 - percentage of decision outcomes exercised by a test suite
- $$= \frac{\text{number of decision outcomes exercised}}{\text{total number of decision outcomes}}$$
- example:
 - program has 2 decision outcomes
 - tests exercise 1 decision outcomes
 - decision coverage = $\frac{1}{2} = 50\%$



62

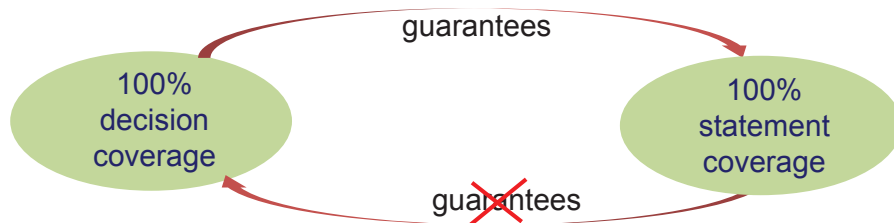
Example of Decision Coverage



63

About Decision Coverage

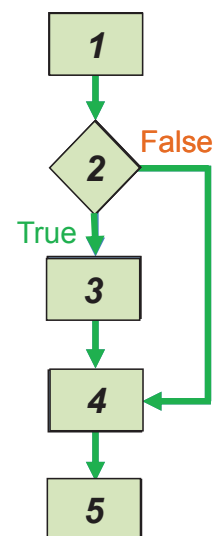
- it is stronger than statement coverage
 - more tests may be needed to achieve 100% decision coverage than are needed for 100% statement coverage
 - therefore, a set of tests that achieve 100% decision coverage may be more effective than a set of tests that achieve 100% statement coverage
 - ▶ i.e. can find more defects (where they exist)



64

How Many Test Cases Required?

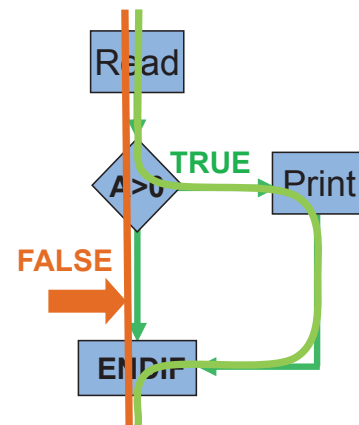
- can calculate the minimum number
 - to achieve statement coverage and decision coverage
- in a control flow diagram
 - for statement coverage
 - ▶ need enough test cases to 'cover' all the 'boxes'
 - for decision coverage
 - ▶ need enough test cases to 'cover' all the 'lines'



65

One Decision, no ELSE

```
Read A
IF A > 0 THEN
  Print "A positive"
ENDIF
```

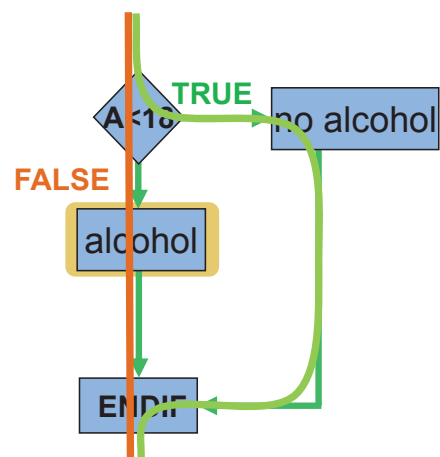


- minimum tests to achieve:
 - ▶ 100% statement coverage: 1
 - ▶ 100% decision coverage: 2

66

One Decision with ELSE

```
IF Age < 18 THEN
  Print "No alcohol"
ELSE
  Print "Have an alcoholic drink"
ENDIF
```



- minimum tests to achieve:
 - ▶ 100% statement coverage: 2
 - ▶ 100% decision coverage: 2

67

Self-draw

Read stock level for item

IF  item in stock THEN

Get from stores

ELSE

Order from supplier

Print "Item back-ordered"

ENDIF

Print "Item processed"

Draw the
diagram
in your
notes

■ minimum tests to achieve:

▶ 100% statement coverage: 2

▶ 100% decision coverage: 2

68

WHILE Loop

Read order line

WHILE more items ordered DO

Check availability

Get from stores

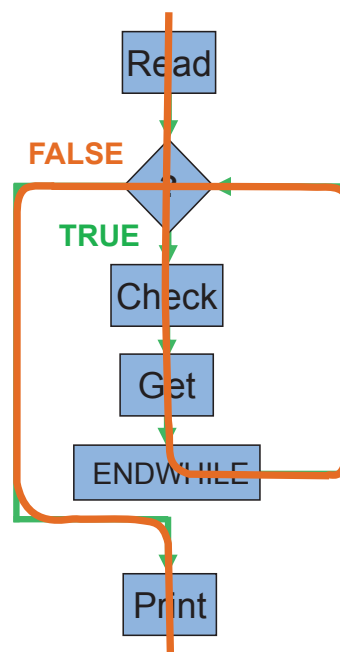
ENDWHILE

Print "Finished processing"

■ minimum tests to achieve:

▶ 100% statement coverage: 1

▶ 100% decision coverage: 1



69

WHILE and IFs

Read Questions

Result = 0

Right = 0

WHILE more Questions

IF Answer = Correct THEN

Right = Right + 1

ENDIF

END WHILE

Result = (Right / Questions)

IF Result > 60% THEN

Print "pass"

ELSE

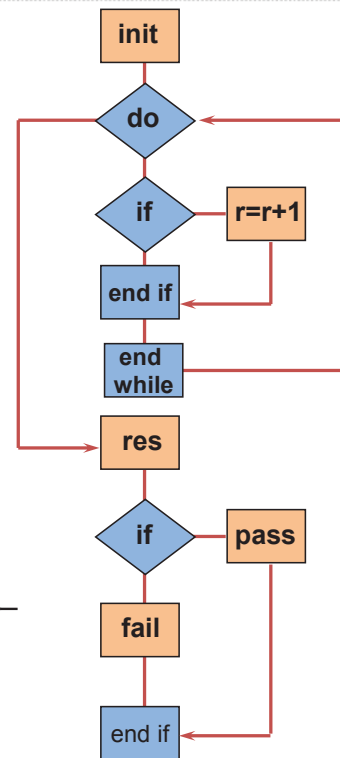
Print "fail"

ENDIF

■ minimum tests to achieve:

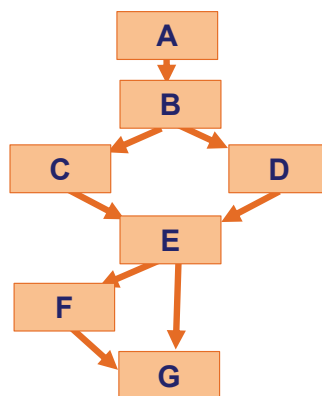
■ 100% statement coverage: 2

■ 100% decision coverage: 2



70

Question: Statement Coverage

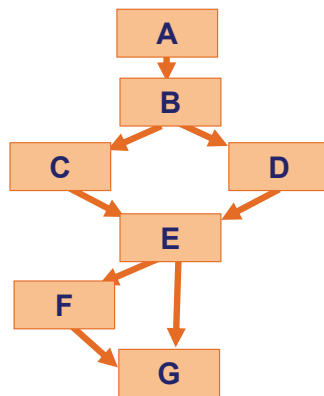


Which test or tests achieve 100% statement coverage?

- a) A,B,C,D,E,F,G
- b) A,B,C,E,G and A,B,C,E,F,G
- c) A,B,C,E,G and A,B,D,E,G
- d) A,B,C,E,F,G and A,B,D,E,F,G

71

Question: Decision Coverage



How much decision coverage has been achieved with the following test cases: A,B,C,E,F,G and A,B,D,E,F,G

- a) 25%
- b) 50%
- c) 75%
- d) 100%

72

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
 - 4.4.1 Statement Testing and Coverage
 - 4.4.2 Decision Testing and Coverage
 - 4.4.3 Other Structure-Based Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

73

Other Levels of Coverage

- there are stronger coverage levels beyond decision coverage
 - e.g. condition combination coverage (multiple conditions), path coverage
 - can be used where more thorough testing is required
- coverage concepts can be applied to higher levels of testing
 - e.g. integration testing: percentage of modules, components and/or classes exercised by test cases
 - e.g. system testing: percentage of menu items

74

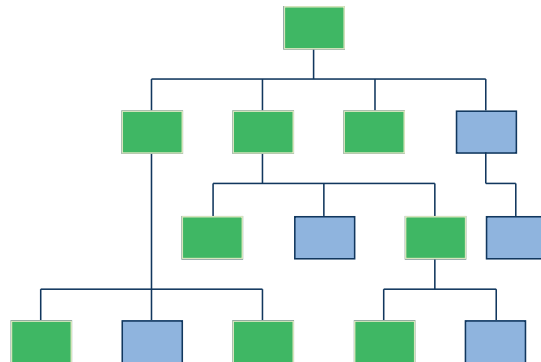
Module / Component Coverage

- percentage of modules exercised by a test suite

$$= \frac{\text{number of modules exercised}}{\text{total number of modules}}$$

- example:

- tests exercise 9 modules
- program has 14 modules
- module coverage = 64%



75

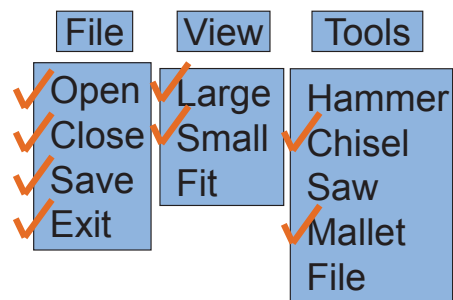
Menu Coverage

- percentage of menu options exercised by a test suite

$$= \frac{\text{number of options exercised}}{\text{total number of options}}$$

- example:

- tests exercise 8 options
- program has 12 menu options
- menu coverage = **66%**



76

Coverage Measurement Tools (Developer)

- objective measure of test coverage
 - tool reports what has and has not been covered by those tests, line by line and summary statistics
- different types of coverage
 - statement, decision, branch, condition, LCSAJ, et al
- intrusive (code is changed)
 - code is instrumented
 - tests are run through the instrumented code
- non-intrusive (code is not changed)
 - this is a sampling technique



77

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

78

Experience-based Techniques

- error guessing
 - based on experience, defect / failure data, or common knowledge of why software fails
 - fault attack - systematic approach to error guessing
 - ▶ list defects/failures based on experience, etc.
 - ▶ design tests to 'attack' these
- exploratory testing
 - a testing approach, not a testing technique

People are creative – at finding defects! Take advantage of their experience, skill & intuition to find more of them!

Testing that is only thorough and systematic is incomplete

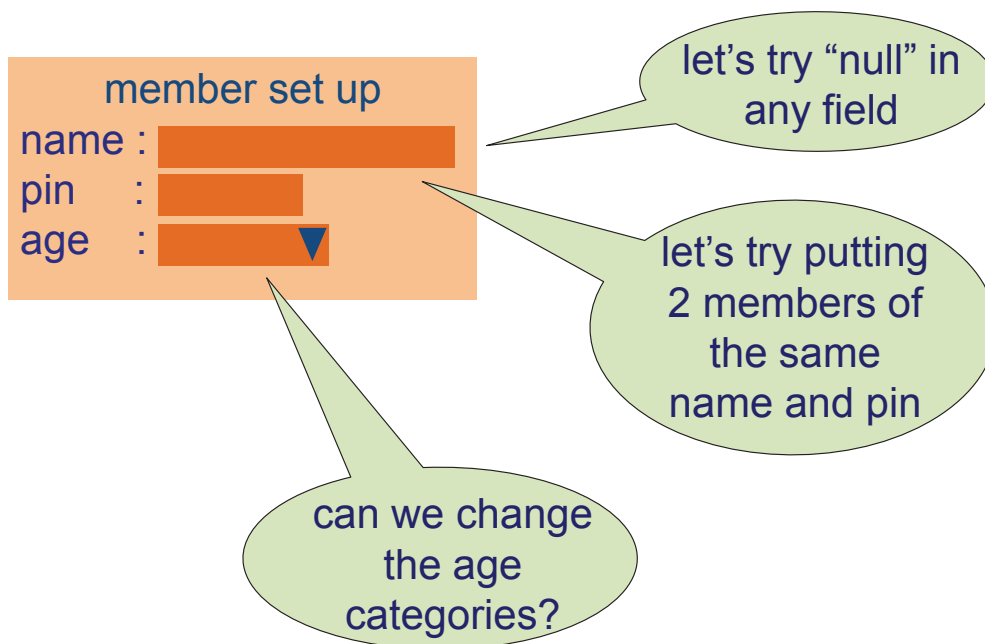
79

Error-Guessing

- perhaps most widely practiced
- always worth including
- can find defects that systematic techniques may miss
- supplements systematic techniques
- consider possible defects using:
 - intuition
 - experience
 - past defects/failures
 - brainstorming
- design test cases to reveal these defects (or confirm their absence)

80

Error-Guessing Example



81

Exploratory Testing

- not a technique in itself – it's an approach
 - can include the use of systematic testing techniques
- it is concurrent
 - test design, test execution, test logging and learning
- based on
 - a test charter containing test objectives / conditions
- carried out
 - within time boxes

82

Key Ideas of Exploratory Testing

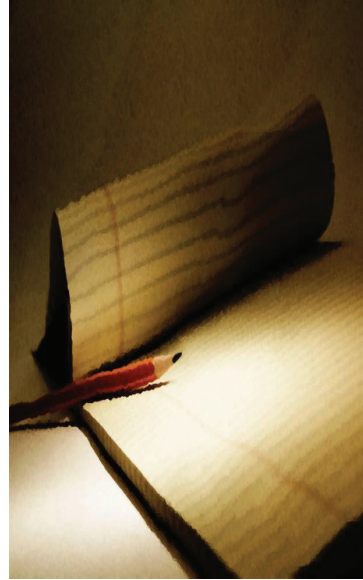
- the results from the tests you design and execute influence the next test you will choose to design and execute
- you build a mental model of the product while you test it
 - this model includes what the product is and how it behaves, and how it's supposed to behave
- you test what you know about, and you are alert for clues about behaviours and aspects of the product that you don't yet know about

Source : James Bach, Satisfice Inc.

83

What Documentation Should We Produce?

- planning
 - test charter, test conditions, test ideas
- execution
 - bug logs, incident reports
- exploring
 - test notes
- post session
 - debrief sheet



84

When to Use Each

use exploratory testing when:

- lack of specifications
- lack of time
- lack of resource
- lack of application knowledge
- goal is to assess weak areas of the system
- also useful to complement formal testing and check test process

use scripted testing when:

- detailed specifications available
- tests are used in automation
- other people required to execute the tests
- it is a legal/company requirement

85

Specification-based Versus Experience-based

- specification-based
 - “the answer” from the specification – the authority
 - techniques are algorithmic (rules to follow)
 - limited to what is in the specification – may miss what isn’t written down
 - more easily taught
- experience-based
 - tester assumes authority
 - more reliance on an individual’s skill
 - brainstorming or collected lists
 - may miss issues because there is no written specification
 - a lateral thinking process, heuristic

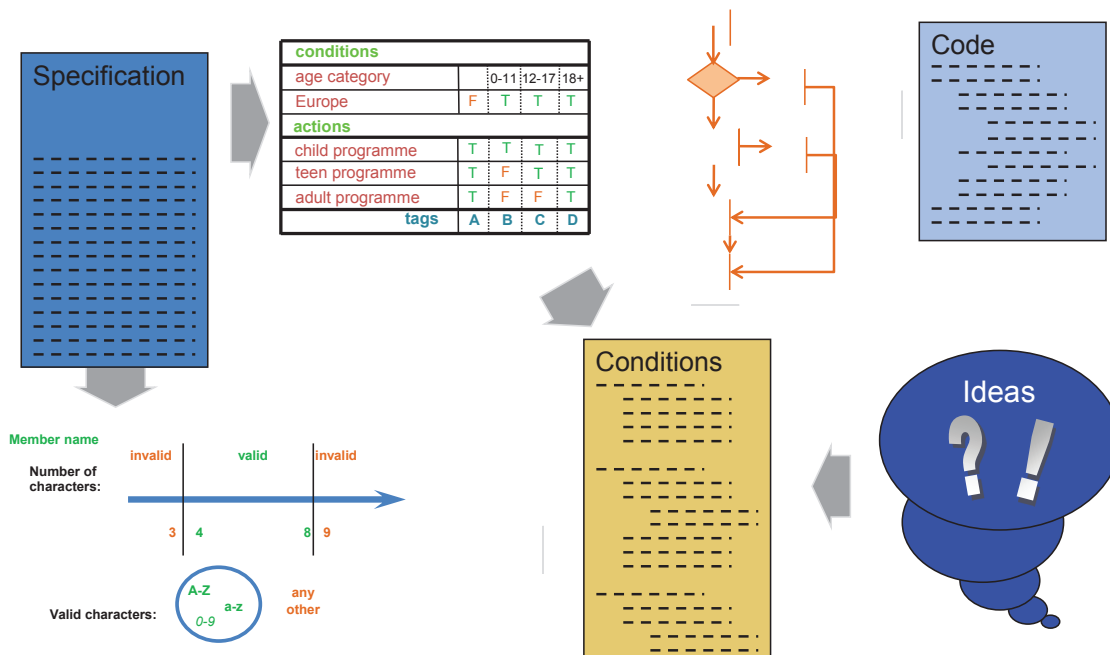
86

Contents

- 4.1 The Test Development Process
- 4.2 Categories of Test Design Techniques
- 4.3 Specification-Based or Black-Box Techniques
- 4.4 Structure-Based or White-Box Techniques
- 4.5 Experience-Based Techniques
- 4.6 Choosing Test Techniques

87

Variety is a Key to Success



88

Which Technique to Use? It Depends!

- external factors
 - risk, customer and contractual requirements, regulatory requirements
 - type of system, time & budget constraints
- internal factors
 - models used (e.g. use cases for requirements), available documentation/specifications
 - tester knowledge & experience of defect types
 - test objective, life cycle model

89

Choosing Appropriate Techniques

	good specs	lack of time	business rules	flow	calculations	coverage	regulatory standards
Equivalence Partitioning	yes	yes	no	no	yes	yes	yes
Boundary Value	yes	no	no	no	yes	yes	yes
Decision Tables	yes	no	yes	no	no	yes	yes
State Transition	yes	no	no	yes	no	yes	yes
Use Cases	yes	no	yes	yes	no	no	yes
Statement Testing	yes	no	yes	no	yes	yes	yes
Decision Testing	yes	no	yes	no	yes	yes	yes
Experience Based	yes	yes	yes	yes	yes	no	no

90

Summary – Key Points

- the test process: how formal depends on context, analyse test basis, test conditions, test cases, test procedures
- categories of test technique: specification-based (black box), structure-based (white box), experience-based (*black box*)
- techniques covered: equivalence partitioning, boundary value analysis, decision tables, state transition, use cases, statement, decision, error guessing, approaches: fault attack, exploratory
- choosing techniques: depends on number of factors: risk, system type, requirements, models, knowledge

91

SESSION 4: TEST DESIGN TECHNIQUES - NOTES

Terms

black-box test design technique, boundary value analysis, code coverage, decision coverage, decision table testing, equivalence partitioning, experience-based test design technique, exploratory testing, (fault) attack, state transition testing, test case specification, test design, test design technique, test execution schedule, test procedure specification, statement coverage, structure-based testing, test script, traceability, use case testing, white-box test design technique.

From the ISTQB Glossary

black-box test design technique: Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

boundary value analysis: A black-box test design technique in which test cases are designed based on boundary values.

code coverage: An analysis method that determines which parts of the software have been executed (covered) by the test suite and which parts have not been executed, e.g., statement coverage, decision coverage or condition coverage.

decision coverage: The percentage of decision outcomes that have been exercised by a test suite. 100% decision coverage implies both 100% branch coverage and 100% statement coverage.

decision table testing: A black-box test design technique in which test cases are designed to execute the combinations of inputs and/or stimuli (causes) shown in a decision table.

equivalence partitioning: A black-box test design technique in which test cases are designed to execute representatives from equivalence partitions. In principle, test cases are designed to cover each partition at least once.

experience-based test design technique: Procedure to derive and/or select test cases based on the tester's experience, knowledge and intuition.

exploratory testing: An informal test design technique where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.

fault attack: Directed and focused attempt to evaluate the quality, especially reliability, of a test object by attempting to force specific failures to occur.

state transition testing: A black-box test design technique in which test cases are designed to execute valid and invalid state transitions.

statement coverage: The percentage of executable statements that have been exercised by a test suite.

structure-based testing: See white-box test design technique.

test case specification: A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item.

test design: The process of transforming general test objectives into tangible test conditions and test cases.

test design technique: Procedure used to derive and/or select test cases.

test execution schedule: A scheme for the execution of test procedures. Note: The test procedures are included in the test execution schedule in their context and in the order in which they are to be executed.

test procedure specification: A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script.

test script: Commonly used to refer to a test procedure specification, especially an automated one.

traceability: The ability to identify related items in documentation and software, such as requirements with associated tests.

use case testing: A black-box test design technique in which test cases are designed to execute scenarios of use cases.

white-box test design technique: Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

Session 3 covered static testing looking at documents and code, but where the code we are interested in is not run. This session looks at dynamic testing, where the software we are interested in is run, by executing tests on the running code.

Now we look at static testing techniques. These techniques are referred to as "static" because the software is not executed; rather the specifications, documentation and source code that comprise the software are examined in varying degrees of detail.

There are two basic types of static testing. One of these is people-based and the other is tool-based. People-based techniques are generally known as "reviews" but there are a variety of different ways in which reviews can be performed. The tool-based techniques examine source code and are known as "static analysis". Both of these basic types are described in the sections below.

4.1 The Test Development Process

Learning Objectives

LO-4.1.1	K2	Differentiate between a test design specification, test case specification and test procedure specification.
LO-4.1.2	K2	Compare the terms test condition, test case and test procedure.
LO-4.1.3	K2	Evaluate the quality of test cases in terms of clear traceability to the requirements and expected results.
LO-4.1.4	K3	Translate test cases into a well-structured test procedure specification at a level of detail relevant to the knowledge of the testers.

Terms

test case specification, test design, test execution schedule, test procedure specification, test script, traceability.

Introduction

Before we can actually execute a test, we need to know what we are trying to test, the exact and specific inputs (and the results that should be produced by those inputs), and how we actually get ready for and run the tests.

In this section we are looking at three things: test conditions, test cases and test procedures (or scripts). Each is specified in its own document, according to IEEE829 (Test Documentation Standard).

Test conditions are documented in a Test Design Specification. We will look at how to choose test conditions and prioritise them.

Test cases are documented in a Test Case Specification. We will look at how to write a good test case, showing clear traceability to the test basis (e.g. requirement specification).

Test procedures are documented (as you may expect) in a Test Procedure Specification (also known as a test script or a manual test script). We will look at how to translate test cases into a test procedure relevant to the knowledge of the tester who will be executing the test, and we will look at how to produce a test execution schedule, using prioritisation and technical and logical dependencies.

Formality of test documentation

Testing may be performed with varying degrees of formality. Very formal testing would have extensive documentation which is well controlled. Very informal testing may have no documentation at all, or only notes kept by individual testers. Most people are probably somewhere in between! The “right” level of formality for you depends on your context: a safety-critical application has very different needs than a one-off application to be used by only a few people for a short time.

The level of formality is also influenced by your organisation – its culture, the people working there, how mature the development process is, how mature the testing process is, and the applicability of safety or regulatory requirements. The thoroughness of your test documentation may also depend on your time constraints: under excessive deadline pressure, keeping good documentation may be compromised.

In this course, we will describe a fairly formal documentation style. If this is not appropriate for you, you might adopt a less formal approach, but you will be aware of how to increase formality if you need to.

Test analysis: Identifying test conditions

Test analysis is the process of looking at something that can be used to derive test information from. This basis for the tests is called the “test basis”. It could be a system requirement or technical specification, for example.

From a testing perspective, we look at the test basis in order to see what could be tested – these are the test conditions. A test condition is simply something that we could test.

For example, if we have a requirements specification, the table of contents can be our initial list of test conditions. If we are looking to measure coverage of code decisions (branches), then the test basis would be the code itself, and the list of test conditions would be the decision outcomes (True and False).

Some people refer to “test requirements” (cf Marick), but this term seems to imply that everything you identify will therefore have to be tested, and this is not true. When identifying test conditions, we want to “throw the net wide” to identify as many as we can, and then we will start being selective about which ones to take forward.

In Session 1 we explained that testing everything is known as exhaustive testing (defined as exercising every combination of inputs and preconditions) and demonstrated that it is an impractical goal. Therefore, as we cannot test everything we have to select a subset of all possible tests. In practice the subset we select may be a very small subset and yet it has to have a high probability of finding most of the defects in a system.

Experience and experiments have shown us that selecting a subset at random is neither very effective nor very efficient (even if it is tool supported). We have to select tests using some intelligent thought process. Test techniques are such thought processes.

A testing technique helps us select a good set of tests from the total number of all possible tests for a given system. Different techniques offer different ways of looking at the software under test, possibly challenging assumptions made about it. Each technique provides a set of rules or guidelines for the tester to follow in identifying test conditions and test cases. Techniques are described in detail later in this chapter.

The test conditions that are chosen will depend on the test strategy or detailed test approach. For example, based on risk, models of the system, likely failures, compliance requirements, expert advice or heuristics.

Test conditions should be able to be linked back to their sources in the test basis – this is called traceability.

Traceability can be either horizontal through all the test documentation for a given test level, (e.g. system testing, from test conditions through test cases to test scripts) or vertical through the layers of development documentation (e.g. from requirements to components).

Why is traceability important? Consider these examples:

Example 1: The requirements for a given function or feature have changed. Some of the fields now have different ranges that can be entered. Which tests were looking at those boundaries? They now need to be changed. How many tests will actually be affected by this change in the requirements? These questions can be answered easily if the requirement can be easily traced to the tests.

Example 2: A set of tests that has run OK in the past has started to have serious problems. What functionality do these tests actually exercise? Traceability between the tests and the requirement being tested enables the functions or features affected to be identified more easily.

Having identified a list of test conditions, it is important to prioritise them, so that the most important test conditions are identified (before a lot of time is spent in designing test cases based on them). It is a good idea to try and think of twice as many test conditions as you need – then you can throw away the less important ones, and you will have a much better set of test conditions to start with!

Note that spending some extra time now, while identifying test conditions, doesn't take very long, as we are only listing things that we could test. This is a good investment of our time – we don't want to spend time implementing poor tests!

Test conditions are documented in the IEEE 829 document called a "Test Design Specification". This document would now probably be better called a "Test Condition Specification", as the contents referred to in the standard are actually test conditions.

Test design: specifying test cases

Test conditions can be rather vague, covering quite a large range of possibilities (e.g. a valid telephone number), or a test condition may be more specific (e.g. the mobile telephone number for a teenaged customer based in Manchester). However when we come to make a test case, now we have to get very specific, in fact we now need exact and detailed specific inputs, not general descriptions. A test condition could be "an existing customer" but the test case input needs to be "Joe Green" where Joe Green already exists on the database.

A test case needs to have input values, of course, but just having some values to input to the system is not a test!

Test cases can be documented as described in IEEE 829 Standard for Test Documentation. Note that the documents described in this standard don't all have to be separate documents. But the standard's list of what needs to be kept track of is a good starting point, even if the test conditions and test cases for a given feature are all kept in one physical document.

One of the most important aspects of a test is that it assesses that the system does what it is supposed to do. If we simply put in some inputs and think "that was fun, I guess the system is probably OK because it didn't crash", then are we actually testing it? We don't think so. You have observed that the system does what the system does – this is not a test.

In order to know what the system should do, we need to have a source of information about the correct behaviour of the system – this is called an "oracle"¹ or a test oracle.

Once a given input value has been chosen, the tester needs to determine what the expected result of entering that input would be, and this is documented as part of the test case.

Expected results would include something displayed on a screen in response to an input, but they also include changes to data and/or states, or any other consequences of the test (e.g. a letter to be printed overnight).

What if we don't decide on the expected results before we run a test? We can still look at what the system produces and would probably notice if something was wildly wrong. However, we would probably not notice small differences in calculations, or results that seemed to look ok (i.e. are plausible). So we would conclude that the test had passed, when in fact the software has not given the correct result.

Ideally expected results should be predicted before the test is run – then your assessment of whether or not the software did the right thing will be more objective.

For a few applications it may not be possible to predict or know exactly what an expected result should be; we can only do a "reasonableness check". In this case we have a "partial oracle" – we know when something is very wrong, but would probably have to accept something that looked reasonable. An example is when a system has been written to calculate something that is actually impossible to calculate manually.

In addition to the expected results, the test case also specifies the conditions that must be in place before the test can be run (the pre-conditions) and any conditions that should apply after the test completes (the post-conditions).

The test case should also say why it exists – i.e. the objective of the test it is part of, or the test conditions that it is exercising (traceability). Test cases can now be prioritised so that the

¹ This has nothing to do with databases or companies that make them! It comes from the ancient Greek Oracle at Delphi, a person who supposedly could predict the future with unerring accuracy.

most important test cases are executed. This may reflect the priorities already established for test conditions, but it could also address priorities of test cases that all cover a given condition, or the priority may be determined by other factors related to the specific test cases, such as a specific input value that has proved troublesome in the past.

Test implementation: specifying test procedures / scripts

The next step is to group the test cases in a sensible way for executing them. For example, a set of simple tests that cover the breadth of the system may form a regression suite, or all of the test that explore the working of a given feature in depth may be grouped to be run together.

Some test cases may need to be run in a particular sequence. For example, a test may create a new customer record, amend that newly created record and then delete it. These tests need to be run in the correct order, or they won't test what they are meant to test.

The document that describes the steps to be taken in running a set of tests (and specifies the executable order of the tests) is called a Test Procedure in IEEE 829, and is often also referred to as a Test Script. It could be called a Manual Test Script for tests that are intended to be run manually rather than using a test execution tool. Test Script is also used to describe the instructions to a test execution tool. An automation script is written in a programming language that the tool can interpret. (This is an Automated Test Procedure.)

The Test Procedures or Test Scripts are then formed into a test execution schedule that specifies which procedures are to be run first – a kind of meta-script. The test schedule would say when a given script should be run and by whom. The schedule could vary depending on newly perceived risks affecting the priority of a script that addresses that risk, for example. The logical and technical dependencies between the scripts would also be taken into account when scheduling the scripts. For example, a regression script may always be the first to be run when a new release of the software arrives, as a smoke test or sanity check.

This is another opportunity to prioritise the tests, to ensure that the best testing is done in the time available.

4.2 Categories of Test Design Techniques

Learning Objectives:

- | | | |
|----------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LO-4.2.1 | K1 | Recall reasons that both specification-based (black-box) and structure-based (white-box) test design techniques are useful and list the common techniques for each. |
| LO-4.2.2 | K2 | Explain the characteristics, commonalities, and differences between specification-based testing, structure-based testing and experience-based testing. |
-

Terms

black-box test design technique, experience-based test design technique, test design technique, white-box test design technique.

Types of Testing Technique

There are many different types of software testing technique, each with its own strengths and weaknesses. Each individual technique is good at finding particular types of fault and relatively poor at finding other types. For example, a technique that explores the upper and lower limits of a single input range is more likely to find boundary value defects than defects associated with combinations of inputs. Similarly, testing performed at different stages in the software development life cycle is going to find different types of defects; component testing is more likely to find coding defects than system design defects.

Each testing technique falls into one of a number of different categories. Broadly speaking there are two main categories, static and dynamic. However, dynamic techniques are

subdivided into three more categories: specification-based (black box, also known as behavioural techniques), structure-based (white box or structural techniques) and experience-based. Specification-based techniques include both functional and non-functional techniques (i.e. quality characteristics). These are summarised below.

Static Testing Techniques

As we saw in Session 3, static testing techniques do not execute the code being examined, and are generally used before any tests are executed on the software. They could be called non-execution techniques. Most static testing techniques can be used to 'test' any form of document including source code, design, functional and requirement specifications. However, 'static analysis' is a tool supported type of static testing that concentrates on testing formal languages and so is most often used to statically 'test' source code.



Specification-Based Testing Techniques (Black Box)

Specification-based testing techniques are also known as 'black-box' or input / output-driven testing techniques because they view the software as a black box with inputs and outputs, but have no knowledge of how it is structured inside the box. In essence, the tester is concentrating on what the software does, not how it does it.



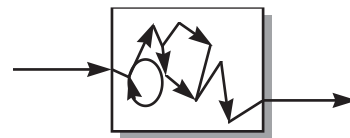
Non-functional aspects (also known as quality characteristics or quality attributes) include performance, usability portability, maintainability, etc. Non-functional testing techniques are concerned with examining how well the system does something, not what it does or how it does it. Techniques to test these non-functional aspects are less procedural and less formalised than those of other categories as the actual tests are more dependent on the type of system, what it does and the resources available for the tests.

How to specify non-functional tests is outside the scope of the syllabus for this course but an approach to doing so is outlined in Tom Gilb's book "Principles of Software Engineering Management", Addison-Wesley, 1988.

Non-functional testing is part of the syllabus and was covered in Session 2 (but techniques for deriving non-functional tests are not covered at the Foundation level).

Structure-Based Testing Techniques (White Box)

Structure-based testing techniques use the internal structure of the software to derive test cases. They are commonly called 'white-box' or 'glass-box' techniques (implying you can see into the system) since they require knowledge of how the software is implemented, that is, how it works. For example, a structural technique may be concerned with exercising loops in the software. Different test cases may be derived to exercise the loop once, twice, and many times. This may be done regardless of the functionality of the software.



Experience-Based Testing Techniques

In experience-based techniques, people's knowledge, skills and background are a prime contributor to the test conditions and test cases. The experience of both technique and business people is important, as they bring different perspectives to the test analysis and design process. Due to previous experience with similar systems, they may have insights into what could go wrong which is very useful for testing.

Note that specification-based and structure-based testing techniques may be used in conjunction with experience-based techniques to leverage the experience of developers, testers and users to determine what should be tested.

Where to apply the different categories of techniques

Specification-based techniques are appropriate at all stages of testing (Component Testing through to Acceptance Testing) where a specification exists. While individual components form part of the structure of a system, when performing Component Testing it is possible to view the component itself as a black box, that is, design test cases based on its specified functionality without regard for its structure.

Structure-based techniques can be used at all stages of testing but are typically used most predominately at Component Testing and Component Integration Testing.

Experience-based techniques are used to complement specification-based and structure-based techniques, and are also used when there is no specification, or if the specification is inadequate or too out of date.

4.3 Specification-Based or Black-Box Techniques

Learning Objectives:

LO-4.3.1	K3	Write test cases from given software models using equivalence partitioning, boundary value analysis, decision tables and state transition diagrams/tables.
LO-4.3.2	K2	Explain the main purpose of each of the four testing techniques, what level and type of testing could use the technique, and how coverage may be measured.
LO-4.3.3	K2	Explain the concept of use case testing and its benefits.

Terms

boundary value analysis, decision table testing, equivalence partitioning, state transition testing, use case testing.

4.3.1 Equivalence Partitioning & Boundary Value Analysis

Equivalence partitioning

Equivalence Partitioning (EP) is a good all-round specification-based black-box technique. It can be applied at any level of testing and is often a good technique to use first. It is a common sense approach to testing, so much so that most testers practise it informally even though they may not realise it. However, while it is better to use the technique informally than not at all, it is much better to use the technique in a formal way to attain the full benefits that it can deliver.

The idea behind the technique is to divide (i.e. partition) a set of test conditions into groups or sets that can be considered the same (i.e. equivalent), hence 'equivalence partitioning'. Equivalence partitions are also known as equivalence classes - the two terms mean exactly the same thing.

The equivalence partitioning technique then requires that we need test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work and so there is no point in testing any of these others. Conversely, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is no point in testing any more in that partition. Of course these are simplifying assumptions that may not always be right but if we write them down, at least it gives other people the chance to challenge the assumptions we have made and hopefully help to identify better partitions.

For example, a savings account in a bank earns a different rate of interest depending on the balance in the account. In order to test the software that calculates the interest due we can identify the ranges of balance values that earns the different rates of interest. For example,

if a balance in the range £0 to £100 has a 3% interest rate, a balance between £100 and £1,000 has a 5% interest rate, and balances of £1,000 and over have a 7% interest rate, we would initially identify three valid equivalence partitions and one invalid partition as shown below.

Invalid partition	Valid (3% interest)		Valid (for 5%)		Valid (for 7%)
£-0.01	£0.00	£100.00	£100.01	£999.99	£1,000.00

When designing the test cases for this software we would ensure that the three valid equivalence partitions are each covered once (and we would also test the invalid partition at least once). So for example, we might choose to calculate the interest on balances of £50, £260 and £1,348. Had we not have identified these partitions it is possible that at least one of them could have been missed at the expense of testing another one several times over.

For example, without thinking about the partitions, a Naïve Tester (NT) might have thought that a good set of tests would be to test every £50. That would give the following tests: £50, £100, £150, £200, £250, ... say up to £750 (then NT would have got tired of it and thought that enough tests had been carried out). But look at what NT has tested: only two out of three partitions! So if the system does not correctly handle a balance of £1000 or more, NT would not have found that defect – so is less effective than Equivalence Partitioning (EP). At the same time, NT has five times more tests (15 tests versus our 3 tests using EP), so it is also much less efficient! This is why we say that using techniques such as EP makes testing both more effective and more efficient.

Note that when we say a partition is “invalid”, it doesn’t mean that it represents a value that shouldn’t be entered by a user. It just means that it is not one of the expected inputs for this particular field. The software should correctly handle values from the invalid partition, by replying with an error message such as “Balance must be at least £0.00”

Note also that the invalid partition may only be invalid in the context of calculating interest payments. An account that is overdrawn will require some different action.

Boundary value analysis

Boundary Value Analysis (BVA) is based on testing at the boundaries between partitions. If you have done “range checking”, you were probably using the boundary value analysis technique, even if you weren’t aware of it. Note that we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions).

As an example, consider a printer that has an input option of the number of copies to be made. This may be from 1 to 99.

Invalid	Valid	Invalid
0	1 99	100

We have taken minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case).

Consider the bank system described in the section on equivalence partitioning.

Invalid partition	Valid (for 3% int)		Valid (for 5%)		Valid (for 7%)
£-0.01	£0.00	£100.00	£100.01	£999.99	£1,000.00

Because the boundary values are defined as those values on the edge of a partition, we have identified the following boundary values: £-0.01 (and invalid boundary value because it is at the end of an invalid partition), £0.00, £100.00, £100.01, £999.99 and £1,000.00.

So by applying boundary value analysis (BVA) we will have six tests for boundary values. Compare what our Naïve Tester (NT) had done: He or she did actually hit one of the boundary values (£100) though it was more by accident than design. So in addition to testing only two thirds of the partitions, NT has only tested one sixth of the boundaries (so will be less effective at finding any boundary defects). If we consider all of our tests for both EP and BVA,

the techniques give us a total of nine tests, compared to the 15 that NT had, so we are still considerably more efficient as well.

Note that in the bank interest example, we have valid partitions next to other valid partitions. If we were to consider an invalid boundary for the 3% interest rate, we have £-0.01, but what about the value just above £100.00? The value of £100.01 is not an invalid boundary, it is actually a valid boundary because it falls into a valid partition. So the partition for 5%, for example, has no invalid boundary values.

What about the boundary values on the other side for the 7% interest rate? (and it is similar for the other end of the invalid partition) What is the maximum value for an account balance?

We could go back to the specification for this to see if a maximum is stated. If so, then we know what our boundary value is. But what if it is not stated in the specification for this item? Can we still test the maximum boundary for the “at least £1000 balance” partition? This is called an Open Boundary, because one of the sides of the partition is left open, i.e. not defined.

Open boundaries are more difficult to test, but there are ways to approach them. Actually the best solution to the problem is to find out what the boundary should be specified as! Other approaches might be to investigate other related areas of the system. For example, the field that holds the account balance figure may be only six figures plus the 2 decimal figures. This would give a maximum account balance of £999,999.99 so we could use that as our maximum boundary value. If we really cannot find anything about what this boundary should be, then we probably need to use an exploratory approach to probe various large values trying to make it fail.

3-Value BVA

You will need to be aware that there are two different “flavours” of Boundary Value Analysis. So far, we have discussed the “2-value” approach. Here we think of the boundary as a dividing line between two things. Hence we have a value on each side of the boundary (but the boundary itself is not a value).

Invalid	Valid	Invalid
0	1 99	100

Looking at the values for our printer example, 0 is in an invalid partition, 1 and 99 are in the valid partition and 100 is in the other invalid partition. So the boundary is between the values of 0 and 1 and between the values of 99 and 100.

The “3-value” approach to Boundary Value Analysis requires a value **“on the boundary and either side of it”**. Now the values of 1 and 99 are regarded as “on” the boundary, so we would have a value less than 1 (i.e. 0) and a value greater than 1 (i.e. 2), and also a value less than 99 (i.e. 98) and a value greater than 99 (i.e. 100).

Invalid	Valid	Invalid
0	1 2 98 99	100

The “on-boundary” values are here considered to be within the valid partition, so the requirement may have been expressed as “Valid values are between 1 and 99 inclusive”.

So which approach is best? If you use the two-value approach together with equivalence partitioning, you are equally effective and slightly more efficient than the three value approach. (We won’t go into the details here but this can be demonstrated.)

Extending EP and BVA

So far, by using EP and BVA we have identified test conditions that could be tested, i.e. partitions and boundary values. We have been looking at applying these techniques to ranges of numbers. However, we can also apply the techniques to other things:

For example, if you are booking a flight, you have a choice of Economy/Coach, Premium Economy, Business or First Class. Each of these is an equivalence partition in its own right

and should be tested, but it doesn't make sense to talk about boundaries for this type of partition, which is a collection of valid things. The invalid partition would be an attempt to type in any other type of flight class (e.g. "Staff"). If this field is implemented using a drop-down list, then it should not be possible to type anything else in, but it is still a good test to try at least once in some drop-down field!

We can apply EP and BVA to all levels of testing. The examples here were at a fairly detailed level probably most appropriate in Component Testing or in detailed testing of a single screen.

For example, we may have three basic configurations which our users can choose from when setting up their systems, with a number of options for each configuration. The basic configurations could be System Administrator, Manager and Customer Liaison. These represent three equivalence partitions that could be tested. We could have serious problems if we forget to test the configuration for the System Administrator, for example.

We have been looking so far at input partitions – i.e. where we identify similarities in the "input space". However, we can also apply these techniques to output partitions. In our bank example, we have three output partitions representing the three different interest rates. In this example, we don't gain much because the output partitions correspond with the input partitions.

Suppose that a customer with more than one account can have an extra 1% interest on this account if they have at least £1000 in it. Now we have two possible output values (7% interest and 8% interest) for the same account balance, so we have identified another test condition (8% interest rate). (We may also have identified that same output condition by looking at customers with more than one account, which is a partition of types of customer.)

Equivalence Partitioning can be applied to different types of input as well. Our examples have concentrated on inputs that would be typed in by a (human) user when using the system. However systems receive input data from other sources as well, such as from another system via some interface – this is also a good place to look for partitions (and boundaries). For example the value of an interface parameter may fall into valid and invalid equivalence partitions. This type of defect is often difficult to find in testing once the interfaces have been joined together, so is particularly useful to apply in integration testing.

Partitions can also be identified when setting up test data. If there are different types of record, your testing would be more representative if you include a data record of each type. The size of a record is also a partition with boundaries, so we could include maximum and minimum size records in the test database.

If you have some "inside knowledge" about how the data is physically organized, you may be able to identify some "hidden boundaries". For example, if an overflow storage block is used if more than 255 characters are entered into a field, the boundary value tests would include 255 and 256 characters in that field.

Design Test Cases

Having identified the conditions that you wish to test, in this case by using equivalence partitioning and boundary value analysis, the next step is to design the test cases. The more test conditions that can be covered in a single test case, the fewer test cases will be needed.

One test case can cover one or more test conditions. Using the bank balance example, our first test could be of a new customer with a balance of £500. This would cover partitions for a balance in the partition from £100.01 to £999.99, and an output partition of a 5% interest rate. We would also be covering other partitions that we haven't discussed yet, for example a valid customer, a new customer, a customer with only one account, etc. All of the partitions covered in this test are valid partitions.

When we come to test invalid partitions, we may try to cover only one test condition per test case. This is because programs typically stop processing input as soon as they encounter the first problem. So if you have an invalid customer name, invalid address, and invalid balance, you may get an error message saying "invalid input" and you don't know whether the software has tested only one invalid input or all of them. (This is also why specific error messages are much better than general ones!)

However, if it is known that the software under test is required to process all input regardless of its validity, then it is sensible to continue as before and design test cases that cover as many invalid conditions in one go as possible. For example, if every invalid field in a form has some red text below it saying that the field is invalid, then you know that every one has been checked, so you have tested all of the error processing in one test case.

In either case, there should be separate test cases covering valid and invalid conditions.

The test cases to cover the boundary conditions are done in a similar way.

Why do both EP and BVA?

Technically, because every boundary is in some partition, if you did only boundary value analysis (BVA) you would also have tested every equivalence partition (EP). However, this approach will cause problems when the value fails – was it only the boundary value that failed or did the whole partition fail? Also by testing only boundaries we would probably not give the users too much confidence as we are using extreme values rather than normal values. The boundaries may be more difficult (and therefore more costly) to set up as well.

For example, in the printer copies example described earlier we identified the following boundary values using the 2-value approach: 0, 1, 99 and 100. Using both Equivalence Partitioning and 2-value Boundary Value Analysis gives the following tests (the EP values are examples):

Invalid		Valid			Invalid	
-24	0	1	13	99	100	144

If we were to test only the boundary values, we could find that only the values of 1 and 99 work successfully. Our assumption is that this means that all values between 1 and 99 will work successful, but there is a small chance that the other values may not work. Using Equivalence Partitioning in conjunction with Boundary Value Analysis (i.e. the value of 13) confirms that an intermediate value also works, so the probability of all of them working is now much higher (though not 100%).

We recommend that you test the partitions separately from boundaries - this means choosing partition values that are NOT boundary values.

What partitions and boundaries you exercise and which first depends on your objectives. If your goal is the most thorough approach, then follow the traditional approach and test valid partitions, then invalid partitions, then valid boundaries and finally invalid boundaries. However if you are under time pressure and cannot test everything (and who isn't), then your objective will help you decide what to test. If you are after user confidence with minimum tests, you may do valid partitions only. If you want to find as many defects as possible as quickly as possible, you may start with invalid boundaries.

4.3.2 Decision Table Testing

Why use decision tables?

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs. However if different combinations of inputs result in different actions being taken, this can be more difficult to show using EP/BVA.

One way to deal with combinations is to use a decision table, which is sometimes also referred to as a “cause-effect” table. The reason for this is that there is an associated logic diagramming technique called “cause-effect graphs” which was sometimes used first to help derive the decision table. However, most people find it more useful just to use the table.

As well as being useful for testing, Decision tables can be used during the design and specification of a system or component to help to specify the effects of different combinations of inputs or system states. They provide a systematic way of stating these complex business rules, which is useful for developers as well as for testers. They can be used in test design whether or not they are used in specifications as they help testers explore the effects of

combinations of different inputs and other software states that must correctly implement business rules based on a set of conditions.

If you begin using decision tables to explore what the business rules are that should be tested, you may find that the analysts and developers find them very helpful and want to begin using them too. Do encourage this, as it will make your job easier in the future.

Testing combinations can be a challenge, as the number of combinations is can often be huge. Testing all combinations is impractical if not impossible. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is not trivial. If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

Decision tables aid the systematic selection of effective test cases and can have the beneficial side-effect of finding problems and ambiguities in the specification. It is a technique that works well in conjunction with equivalence partitioning. The combination of conditions explored may be combinations of equivalence partitions.

In addition to decision tables, there are other techniques that deal with testing combinations of things: pairwise testing and orthogonal arrays. These are described in Lee Copeland: A Practitioner's Guide to Software Test Design, Artech House, 2004.

Using decision tables for test design

The first task is to identify a suitable function or subsystem that has a behaviour which reacts according to a combination of inputs or events. The behaviour of interest must not be too extensive (i.e. should not contain too many inputs) otherwise the number of combinations will become cumbersome and difficult to manage. It is better to deal with large numbers of conditions by dividing them into subsets and dealing with each subset one at a time.

For example, consider a system that compiles a CD comprising all of the files that are to be printed. All files on the CD must be in PDF format so files that are already in PDF format can be copied to the CD whereas files of other formats have to have a PDF format version made. The conditions of interest are "File to be printed?" and "PDF format?" The actions that can result from various combinations of these conditions are "Copy" and "Make PDF". These are shown in the decision table below.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
File to be printed?	Y	Y	N	N
PDF format?	Y	N	Y	N
Actions				
Copy PDF to CD	Y	Y	N	N
Make PDF	N	Y	N	N

As can be seen, the conditions and actions are listed in the left hand column. All the other columns in the decision table each represent a separate rule, one for each combination of conditions. We may choose to test each rule/combination and if there are only a few this will usually be the case. However, if the number of rules/combinations is large we are more likely to sample them by selecting a rich subset for testing.

If we are applying this technique thoroughly, we would have one test for each column of our decision table. The advantage of doing this is that we may test a combination of things that otherwise we might not have tested.

There may also be many different actions. The decision table shows which actions apply to each combination of conditions.

In this example above both the conditions are binary, i.e. they have only two possible values: True or False (or if you prefer: Yes or No). Often it is the case that conditions are more complex, having potentially many possible values. Where this is the case the number of combinations is likely to be huge.

Dealing with large numbers of combinations

As we have said previously, when the number of rules/combinations is large we are more likely to sample them by selecting a rich subset for testing. This is because it will become impractical, if not impossible, to test every combination, even if we really want to. This size of the subset will be governed by the relative importance of this part of the system compared with other parts of the system. If the conditions we are exploring are a critical part of the system then we will be persuaded to spend the time testing more of the combinations.

Having only two values (T/F) means that the variable can be referred to as “Boolean”. A table with only Boolean values may be called a “limited entry decision table”, since the entries are limited to the Boolean type. But sometimes it makes more sense to use the table in a more flexible way. Take the following example. Here we have added a third condition to the two from the previous example. This new condition is a code that can have three different values (1, 2 or 3). The effect of this is that it causes us to repeat the original block of combinations twice for the other code values.

Conditions	Original				Repeated block 1				Repeated block 2			
Code = 1, 2 or 3	1	1	1	1	2	2	2	2	3	3	3	3
File to be printed?	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
PDF format?	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

Given that there are only twelve combinations in total it is both practical and (probably) useful to document them all in the table. However, if the ‘Code’ condition could have 30 different values then there would be little value in documenting them all. One possible approach we could take is shown below.

Conditions	1	2	3	4		117	118	119	120
Code = 1 to 30 inclusive	1	1	1	1		30	30	30	30
File to be printed?	Y	Y	N	N		Y	Y	N	N
PDF format?	Y	N	Y	N		Y	N	Y	N

The advantage of doing this is that it highlights the total number of combinations and the fact that they comprise 30 copies of 4 basic combinations. Of course we do not have to present the decision table in this way to highlight these things. If doing so is likely to reduce the risk of people neglecting important combinations or failing to recognise the number of combinations in the first place, then it is worth doing.

The whole purpose of producing a decision table is to help us recognise the total number of combinations and then to help us make an informed decision about which ones we will test. For example, consider our earlier example with 12 combinations. Let us assume that we do not have sufficient time to test each one (or at least we deem it not to be important enough to do so). Let us assume we would like to test just three combinations as, given that the ‘Code’ condition can take three values, this would permit us to test each one once. Selecting three combinations without due consideration we may select the first one in each block of four so we end up with combinations 1, 5 and 9. This will indeed give us three test cases covering three different Code values but also results in the other two conditions being unchanged (they will only be tested in the Yes/Yes combination).

Conditions	1	2	3	4	5	6	7	8	9	10	11	12
Code = 1, 2 or 3	1	1	1	1	2	2	2	2	3	3	3	3
File to be printed?	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
PDF format?	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

A better scheme might be to select combinations 1, 6 and 11. This subset ensures that we have different combinations of the other two conditions for each test case, specifically Yes/No and No/Yes. This may be satisfactory but arguably could be improved further by substituting one of the Yes/No, No/Yes combinations for a No/No combination, thus combinations 1, 6 and 12 could be a final choice.

The difference between the subset 1, 6, 11 and 1, 6, 12 is slight but could be significant. Ultimately a better knowledge of the application may help us decide which is the more important subset. Yet another possibility would be to argue a case for testing four combinations such as 1, 6, 11 and 12.

4.3.3 State Transition Testing

State transition testing is used where some aspect of the system can be described in what is called a “finite state machine”. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the “machine”. This is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

State diagram: a diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another (i.e. from a state transition).

State table: a grid showing the resulting transitions for each state combined with each possible event, showing both valid and invalid transitions.

For example, if you request to withdraw £100 from a bank ATM, you may be given cash. Later you may make exactly the same request but be refused the money (because your balance is insufficient). This later refusal is because the state of your bank account had changed from having sufficient funds to cover the withdrawal to having insufficient funds. The transaction that caused your account to change its state was probably the earlier withdrawal. Another example is a word processor. If a document is open, you are able to close it. If no document is open, then “Close” is not available. After you choose “Close” once, you cannot choose it again for the same document unless you open that document. A document thus has two states: open and closed.

A state transition model has four basic parts:

- the states that the software may occupy (open/closed or funded/insufficient funds);
- the transitions from one state to another (not all transitions are allowed);
- the events that cause a transition (withdrawing money, closing a file);
- the actions that result from a transition (an error message, or being given your cash).

Note that a transition does not need to change to a different state, it could stay in the same state. In fact, trying to input an invalid input would be likely to produce an error message as the action, but the transition would be back to the same state the system was in before.

Test conditions can be derived from the model (state graph or state chart) in various ways. Each state can be noted as a test condition, as can each transition. We can also consider transition pairs and triples and so on. More correctly, single transitions are known as 0-switch coverage, transition pairs as 1-switch coverage, transition triples as 2-switch coverage, etc. Deriving test cases from the state transition model is a black box approach. Measuring how much you have tested (covered) is getting close to a white box perspective. However, state transition testing is generally regarded as a black box technique.

One of the beauties of this technique is that the model can be as detailed or as abstract as you need it to be. Where a part of the system is more important (that is, requires more testing) a great depth of detail can be modelled. Where the system is less important (require less testing) the model can use a single state to signify what would otherwise be a series of different states.

Deriving tests only from a state graph (or state chart) may omit the negative tests, where we could try to generate invalid transitions. In order to see the total number of combinations of states and transitions, both valid and invalid, a state table can be used. This lists all the states down one side of the table and all the events that cause transitions along the top (or vice versa). Each cell then represents a state / event pair. The content of each cell indicates to which state the system will move when the corresponding event occurs whilst the associated

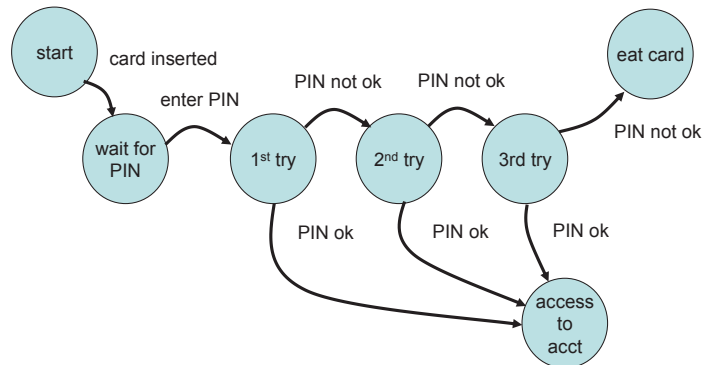
state. This will include possible erroneous events – events that are not expected to happen in certain states. This usually makes good negative test conditions.

Deriving test cases from the state transition model is a black box approach. Measuring how much you have tested (covered) is getting close to a white box perspective. However, state transition testing

Here is an example

Three tries for PIN

output.



State Transition Graph modelling a possible set of states and transitions when a user tries to gain access to their bank account.

This is the state diagram. We have shown seven states, but only four possible actions. We have not specified all of the possible transitions here – there would also be a time-out from the “wait for PIN” and the three tries to go back to the start state and probably eject the card. There would also be a transition from the “eat card” state back to the start state. We have not specified all the possible actions either – there would be a “cancel” option from the “wait for PIN” and the three tries which would also go back to the start state and eject the card. The “access to account” state would be the beginning of another state diagram showing the valid transactions that could now be performed on the account.

However this gives us information on which to design some tests.

A sensible first test case would be the normal situation, where the correct PIN is entered the first time. A second test could be to enter an incorrect PIN each time, so that the system eats the card. The tests for the 2nd and 3rd try are probably less important, but if we want to test every transition, then we will need the two other tests for those as well.

4.3.4 Use Case Testing

Use Case Testing is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish. They were invented by Ivar Jacobsen and published in his book Object-Oriented Software Engineering: A Use Case Driven Approach.

A use case is a description of a particular use of the system by an actor (a user of the system). Each use case describes the interactions the actor has with the system in order to achieve a specific task (or at least produces something of value to the user). Actors are generally people but they may also be other systems. Use cases are described as a sequence of steps that describe the interactions between the actor and the system. If a use case involves multiple actors, one actor must be identified as the principle actor and the use case is written from the perspective of this person or system.

Use cases are defined in terms of the user, not the system, describing what the user does and what the user sees rather than what inputs the system expects and what the system outputs. They use the language and terms of the business rather than technical terms. Use cases may be described at an abstract level giving a business view. This will be technology-free (no technical terminology) but using business terminology. Use cases may also be described at the system level, focused on the system functionality, though this is not as common.

Use cases can help to uncover integration defects, that is, defects caused by the incorrect interaction between different components. They can serve as a foundation for designing test cases at the system and acceptance testing levels.

Use cases describe the process flows through a system based on its most likely use. This makes the test cases derived from use cases particularly good for finding defects in the real-world use of the system (i.e. the defects that the users are most likely to come across when first using the system). Each use case usually has a main (or most likely) scenario and sometimes alternative scenarios (covering, for example, special cases or exceptional conditions). Each use case must specify any preconditions that need to be met for the use case to work. Use cases must also specify post-conditions that are observable results and a description of the final state of the system after the use case has been executed successfully.

System requirements can be specified as a set of use cases or may include use cases to complement a requirement specification. This approach can make it easier to involve the users in the requirements gathering and definition process.

4.4 Structure-Based or White-Box Techniques

Learning Objectives:

LO-4.4.1	K2	Describe the concept and value of code coverage.
LO-4.4.2	K2	Explain the concepts of statement and decision coverage, and give reasons why these concepts can also be used at levels other than component testing (e.g. on business procedures at system level).
LO-4.4.3	K3	Write test cases from given control flows using statement and decision test design techniques.
LO-4.4.4	K4	Assess statement and decision coverage for completeness with respect to defined exit criteria.

Terms

code coverage, decision coverage, statement coverage, structure-based testing.

Introduction

Structure-based testing techniques are normally used after an initial set of tests has been derived using specification-based techniques. They are most often used to measure "coverage" - how much of the structure has been exercised or covered by a set of tests.

Coverage measurement is best done using tools, and there are a number of such tools on the market. These tools can help to increase productivity and quality. They increase quality by ensuring that more structural aspects are tested, so defects on those structural paths can be found. They increase productivity and efficiency by highlighting tests that may be redundant, i.e. testing the same structure as other tests (although this is not necessarily a bad thing!)

What are Coverage Techniques?

Coverage techniques serve two purposes: test measurement and test case design. They are often used in the first instance to assess the amount of testing performed by tests derived from specification-based techniques. They are then used to design additional tests with the aim of increasing the test coverage.

Coverage techniques are a good way of generating additional test cases that are different from existing tests and in any case they help ensure breadth of testing in the sense that test cases that achieve 100% coverage in any measure will be exercising all parts of the software from the point of view of the items being covered. There is also danger in these techniques. 100% coverage does not mean 100% tested. Coverage techniques measure only one dimension of a multi-dimension concept. Two different test cases may achieve exactly the same coverage but the input data of one may find an error that the input data of the other

doesn't. Furthermore, coverage techniques measure coverage of the software code that has been written, they cannot say anything about the software that has not been written. If a specified function has not been implemented, specification-based testing techniques will reveal this. If a function was omitted from the specification, then experience-based techniques may find this.

In common with all structure-based testing techniques, code coverage techniques are best used on areas of software code where more thorough testing is required. Safety critical code, code that is vital to the correct operation of a system, and complex pieces of code are all examples of where structure-based techniques are particularly worth applying. They should always be used in addition to specification-based and experience-based testing techniques rather than as an alternative to them.

Types of Coverage

Test coverage can be measured based on a number of different structural elements in software.

The simplest coverage measure is statement coverage, which measures the percentage of executable statements exercised by a set of tests, compared to all executable statements in the software under test. In fact, all coverage techniques yield a result which is the number of elements covered expressed as a percentage of the total number of elements.

Besides statement coverage, there are number of different types of control flow coverage techniques most of which are tool supported. These include decision coverage, branch coverage, LCSAJ (linear code sequence and jump) coverage, condition coverage and condition combination coverage. Any representation of a system is in effect a model against which coverage may be assessed. Call tree coverage is another example for which tool support is commercially available. Data flow coverage techniques include definitions, uses, and definition-use pairs.

Another popular, but often misunderstood, coverage measure is path coverage. Path coverage is usually taken to mean branch or decision coverage because both these techniques seek to cover 100% of the 'paths' through the code. However, strictly speaking for any code that contains a loop, path coverage is impossible since a path that travels round the loop say 3 times is different from the path that travels round the same loop 4 times. This is true even if the rest of the paths are identical. So if it is possible to travel round the loop an unlimited number of times then there are an unlimited number of paths through that piece of code. For this reason it is more correct to talk about 'independent path segment coverage' though the shorter term 'path coverage' is frequently used.

Other, more specific, coverage measures include things like database structural elements (records, fields, and sub-fields) and files. State transition coverage is also possible. It is worth checking for any new tools, as the test tool market can develop quite rapidly.

How to Measure Coverage

For most practical purposes coverage measurement is something that requires tool support. However, knowledge of the steps typically taken to measure coverage is useful in understanding the relative merits of each technique.

1. Decide on the structural element to be used.
2. Count the structural elements.
3. Instrument the code.
4. Run the tests for which coverage measurement is required.
5. Using the output from the instrumentation, determine the percentage of elements exercised.

Instrumenting the code (step 3) implies inserting code along-side each structural element in order to record that the associated structural element has been exercised. Determining the actual coverage measure (step 5) is then a matter of analysing the recorded information. Our

example assumes an “intrusive” coverage measurement tool that does actually alter the code by inserting the instrumentation.

If you are aiming for a given level of coverage (say 95%) but you have not reached your target (say only 87%), then additional test cases have to be designed with the aim of exercising some or all of the structural elements not yet reached. This is structure-based test design. These are then run through the instrumented code and a new coverage measure is calculated. This is repeated until the required coverage measure is achieved (or until you decide that your goal was too ambitious!). Ideally all the tests ought to be run again on the un-instrumented code.

The structure-based techniques are normally used first to measure coverage, then to design tests to extend coverage if needed.

4.4.1 Statement Testing and Coverage

Statement coverage is the number of executable statements exercised by a test or test suite. This is calculated by:

$$\text{Statement Coverage} = \frac{\text{Number_of_statements_exercised}}{\text{Total_number_of_statements}} \times 100\%$$

Typical ad hoc testing achieves 60% to 75% statement coverage.

4.4.2 Decision Testing and Coverage

Decision coverage is the number of decisions exercised by a test or test suite. This is calculated by:

$$\text{Decision Coverage} = \frac{\text{Number_of_decisions_exercised}}{\text{Total_number_of_decisions}} \times 100\%$$

Typical ad hoc testing achieves 40% to 60% decision coverage. Decision coverage is stronger than statement coverage since it may require more test cases to achieve the same measure. For example, consider the code segment shown below.

```
if a > b
    c = 0
endif
```

To achieve 100% statement coverage of this code segment just one test case is required which ensures variable ‘a’ contains a value that is greater than the value of variable ‘b’. However, decision coverage requires each decision to have had both a true and false outcome. Therefore, to achieve 100% decision coverage, a second test case is necessary. This will ensure that the decision statement ‘if a > b’ has a false outcome.

Note that 100% decision coverage guarantees 100% statement coverage.

Branch coverage is closely related to decision coverage, but at 100% coverage they give exactly the same results. (Decision coverage measures the coverage of conditional branches; branch coverage measures the coverage of both conditional and unconditional branches.)

4.4.3 Other Structure-Based Techniques

There are many other structure-based techniques that can be used to achieve testing to different degrees of thoroughness. Some techniques are stronger (require more tests to achieve 100% coverage and therefore, have a greater chance of detecting defects) and others are weaker. Stronger techniques include Condition Testing and Multiple Condition Testing. Weaker techniques include Module Testing and Class Testing.

Using these structure-based techniques and measuring the structural coverage achieved is most often not practical without the aid of a Coverage Measurement Tool. Coverage Measurement tools are described in Session 6.

4.5 Experience-Based Techniques

Learning Objectives:

- | | | |
|----------|----|----------------------------------------------------------------------------------------------------------|
| LO-4.5.1 | K1 | Recall reasons for writing test cases based on intuition, experience and knowledge about common defects. |
| LO-4.5.2 | K2 | Compare experienced-based techniques with specification-based testing techniques. |
-

Terms

exploratory testing, (fault) attack.

Although it is true that testing should be rigorous, thorough and systematic, this is not all there is to testing. There is a definite role for non-systematic techniques.

4.5.1 Error-Guessing

Error guessing is a technique that should always be used after other more formal techniques have been applied. The success of error guessing is very much dependent on the skill of the tester, as good testers know where the defects are most likely to lurk. Some people seem to be naturally good at testing and others are good testers because they have a lot of experience either as a tester or working with a particular system and so are able to pin point its weaknesses. This is why an error guessing approach is best used after more formal techniques have been applied. In using other techniques the tester is likely to gain a better understanding of the system, what it does and how it works. With a better understanding anyone is likely to be more able to think of ways in which the system may not work properly.

There are no rules for error guessing. The tester is encouraged to think of situations in which the software may not be able to cope. Typical conditions to try include divide by zero, blank (or no) input, empty files and the wrong kind of data (e.g. alphabetic characters where numeric are required). If anyone ever says of a system or the environment in which it is to operate "That could never happen", it might be a good idea to test that condition as such assumptions about what will and will not happen in the live environment are often the cause of failures. A structured approach to the error guessing technique is to list possible defects or failures and to design tests that attempt to reproduce them. These defect and failure lists can be built based on the tester's own experience or that of other people, available defect and failure data, and from common knowledge about why software fails.

Error guessing is sometimes known by other names including experience-driven testing and heuristic testing.

4.5.2 Exploratory Testing

Exploratory testing is a hands-on approach in which testers are involved in minimal planning and maximum test execution. The planning involves the creation of a test charter, a short declaration of the scope of a short (1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used.

The test design and test execution activities are performed in parallel typically without documenting the test conditions, test cases or test scripts. This does not mean that other, more formal testing techniques will not be used. For example, the tester may decide to use Boundary Value Analysis but will think through and test the most important boundary values without writing them down.

Test logging is undertaken as test execution is performed, documenting the key aspects of what is tested, any defects found and any thoughts about possible further testing. A key aspect of exploratory testing is learning: learning by the tester about the software, its use, its strengths and weaknesses. As its name implies, exploratory testing is about exploring, finding out about the software, what it does, what it doesn't do, what works and what doesn't work. The tester is constantly making decisions about what to test next and where to spend the (limited) time.

This is an approach that is most useful when there are no or poor specifications and when time is severely limited. It can also serve to complement other, more formal testing, helping to establish greater confidence in the software. In this way, exploratory testing can be used as a check on the formal test process by helping to ensure that the most serious defects have been found.

4.5.3 Other Experience-Based Techniques

(The information given in this shaded section provides additional details not given in the syllabus and is therefore not examinable.)

Trial and Error / Ad Hoc

Many people confuse ad hoc testing with error guessing. Ad hoc testing is unplanned and usually done before rigorous testing. It is good as a quick 'test readiness' assessment. In other words, perform a few things on the latest version of software to determine if it ready for more thorough testing.

Lateral Testing

Lateral testing is an approach to non-systematic testing that uses lateral thinking techniques. Lateral thinking employs alternative methods of thinking. Rather than progress in logical manner from one thought to another, the aim is to make a leap into very different areas of thought.

User Testing

Asking users to "try out" the new software is a form of non-systematic testing. This should only be done once the software is known to be reasonably stable, otherwise the users will lose confidence in the software.

Unscripted Testing

Unscripted testing means testing without a specification of pre-planned tests.

4.6 Choosing Test Techniques

Learning Objective:

LO-4.6.1 K2 Classify test design techniques according to their fitness to a given context, for the test basis, respective models and software characteristics.

The choice of which test techniques to use depends on a number of factors, including the type of system, regulatory standards, customer or contractual requirements, level of risk, type of risk, test objective, documentation available, knowledge of the testers, time and budget, development life cycle, use case models and previous experience of types of defects found.

Some techniques are more applicable to certain situations and test levels; others are applicable to all test levels.

This session has covered the most popular and commonly used software testing techniques. There are many others that fall outside the scope of this course. With so many testing techniques to choose from how are testers to decide which ones to use?

Perhaps the single most important thing to understand is that the best testing technique is no single testing technique! Because each testing technique is good at finding one specific class of defect, using just one technique will help ensure that many (perhaps most but not all) defects of that particular class are found. Unfortunately, it may also help to ensure that many defects of other classes are missed! Using a variety of techniques will therefore help ensure that a variety of defects are found, resulting in more effective testing.

So how can we choose the most appropriate testing techniques to use? The decision will be based on a number of factors, both internal and external. These are described below.

Internal Factors

Models used – since testing techniques are based on models, the models available (i.e. developed and used during the specification, design and implementation of the system) will to some extent govern which testing techniques can be used.

Tester knowledge/experience – how much testers know about the system and about testing techniques will clearly influence their choice of testing techniques. This knowledge will in itself be influenced by their experience of testing and of the system under test.

Likely defects – knowledge of the likely defects will be very helpful in making choosing testing techniques (since each technique is good at finding a particular type of defect). This knowledge can be gained through experience of testing previous version of the system and previous levels of testing on the current version.

Test objective – if the test objective is simply to gain confidence that the software will cope with typical operational use then use cases would sensibly be used. If the objective is for very thorough testing then more rigorous and detailed techniques (including structure-based techniques) should be chosen.

Documentation – whether or not documentation (specification) exist and whether or not they are up-to-date will affect the choice of testing techniques. The content and style of the documentation will also influence the choice of techniques (for example, if decision tables or state graphs have been used then the associated test techniques should be used).

Life cycle model – a sequential lifecycle model will lend itself to the use of more formal techniques whereas an iterative lifecycle model may better suite an exploratory testing approach.

External Factors

Risk – the greater the risk (e.g. safety critical systems) the greater the need for more thorough and more formal testing.

Customer/contractual requirements – sometimes contracts specify particular testing techniques to use (most commonly statement or branch testing).

Type of system – the type of system (e.g. embedded, graphical, financial, etc.) will influence the choice of techniques. For example, a financial application involving many calculations would benefit from Boundary Value Analysis.

Regulatory requirements – some industries have regulatory standards or guidelines that can govern the testing techniques used. For example, the aircraft industry requires the use of Equivalence Partitioning, Boundary Value Analysis and State Transition testing for high integrity systems together with Statement, Decision or Modified Condition Decision Coverage depending on the level of integrity.

Time and budget – ultimately how much time there is available will always affect the choice of testing techniques. When more time is available we can afford to select more techniques and when time is severely limited we will be limited to those that we know have a good chance of helping us find just the most important defects.

Session 5

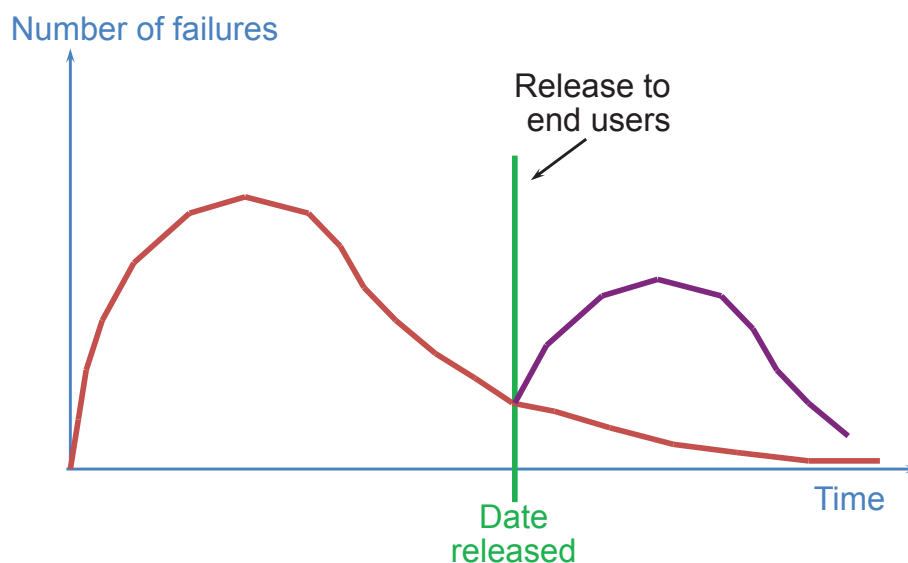
Test Management

Contents

- 5.1 Test Organisation
 - 5.1.1 Test Organisation and Independence
 - 5.1.2 Tasks of the Test Leader and Tester
- 5.2 Test Planning and Estimation
- 5.3 Test Progress Monitoring and Control
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

2

Failure Rate



3

Test Organisation and Independence

- levels of independence*
 - only developers test their own code
 - ▶ not independent
 - developers test each other's code (buddy)
 - tester within the development team
 - test team reporting to Project Manager or higher
 - testers from business and IT
 - test specialist e.g. certification, security
 - tester from an outside organisation (3rd party/outsourced testers)

* as first shown in Session 1

4

Examples of Independence

- large, complex or safety-critical
 - multiple levels of testing
 - independent testers at some (or all) levels
 - ▶ may define test processes and rules
 - testing by developers at lower levels
 - ▶ lack of objectivity could limit effectiveness
- internal application
 - informal component test by developers
 - independent acceptance test by internal users



5

Benefits and Drawbacks of Independence

BENEFITS

- independent testers see different defects / failures
- unbiased / objective assessment
- can verify assumptions made during specification and building of the system
- expertise established in testing

DRAWBACKS

- isolation from development team
- bottleneck as last checkpoint
- developers lose a sense of responsibility for quality
- clearing of incident reports can be slower

6

Contents

5.1 Test Organisation

5.1.1 Test Organisation and Independence

5.1.2 Tasks of the Test Leader and Tester

5.2 Test Planning and Estimation

5.3 Test Progress Monitoring and Control

5.4 Configuration Management

5.5 Risk and Testing

5.6 Incident Management

7

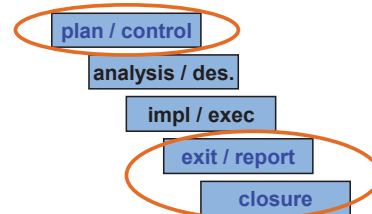
Who Does Testing?

- testing tasks may be done by people in specific testing roles:
 - **test leader**
 - ▶ also known as test manager or test coordinator
 - separate roles (i.e. different people) on large projects
 - **testers**
 - ▶ on separate test team
 - ▶ within development team



8

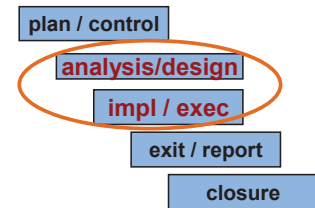
Test Leader Tasks



- *coordinate* strategy and plans to project manager
- *write/review* policy and strategy
- *contribute* testing perspective to project
- *plan* the testing & adapt plans as needed
- *initiate* testing tasks
- *control* testing based on monitoring results
- *establish* configuration management
- *introduce* suitable metrics
- *decide* approach to automation, select tools
- *organise* any training
- *decide* on test environment
- *schedule* tests
- *complete* test summary report

9

Tester Tasks



- *review* and contribute to test plan
- *analyse* requirements and specifications for testability
- *write* test specification
- *set up* test environment
- *prepare* test data
- *run* tests and log results
- *use* tools to log faults, results
- *use* automation tool if required
- *test* non – functional characteristics
- *review* tests written by the team



10

Who Else Does Testing?

- in addition to the “obvious” roles
 - *test leader and testers*
- testing tasks may be done by:
 - *other roles within the organisation*
 - ▶ project manager
 - ▶ quality manager
 - ▶ developer
 - ▶ business / domain expert
 - ▶ infrastructure / IT operations
 - *e.g. for small projects or small companies*



11

Contents

5.1 Test Organisation

5.2 Test Planning and Estimation

5.2.1 Test Planning

5.2.2 Test Planning Activities

5.2.3 Entry Criteria

5.2.4 Exit Criteria

5.2.5 Test Estimation Проценка

5.2.6 Test Strategy, Test Approach

5.3 Test Progress Monitoring and Control

5.4 Configuration Management

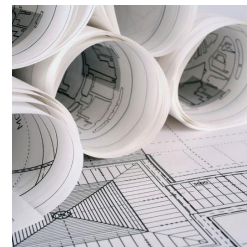
5.5 Risk and Testing

5.6 Incident Management

12

Test Planning

- what is the purpose of a test plan?
 - who does it communicate to?
 - why is it a good idea to have one?
- what information should be in a test plan?
 - do you have a standard/template for a test plan?
 - do you think anything is missing?



Test plans are vital to think about who, what, when and how for testing on the project. Information should be sent to the appropriate people on the project so that they can provide feedback.

13

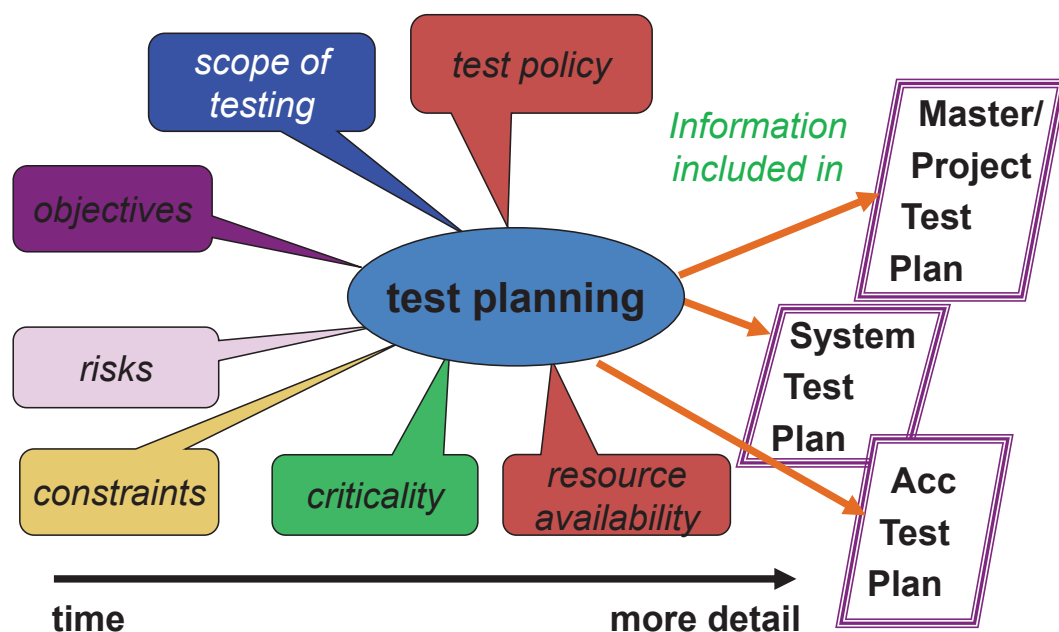
Project / Master Test Plan

- apply company testing approach or strategy
 - test levels, entry and exit criteria
- document any exceptions
 - reasons for non-compliance must be given
- consistent approach to testing is encouraged
- builds a framework for other test plans
 - e.g. test plans to address several test levels or stages

Source: ANSI/IEEE Std 829-1998, Test Documentation Standard

14

Test Planning Influences



15

Test Plan Headings 1

- 1. test plan identifier
- 2. introduction
 - management summary, references to other documents (e.g. project plan)
- 3. test items
 - test items – what is to be tested (e.g. software, documents) versions, how transmitted, references to other documentation
- 4. features to be tested
 - identify functionality and test design specification / techniques
- 5. features not to be tested
 - reasons for exclusion

16

Test Plan Headings 2

- 6. approach (to testing)
 - e.g. types of activities, techniques and tools
 - identify constraints (environment, staff, deadlines)
- 7. item pass/fail criteria (for what is tested)
 - ▶ e.g. number of known defects outstanding by severity
- 8. suspension criteria and resumption criteria
 - for all or parts of testing activities
 - which activities must be repeated on resumption

17

Test Plan Headings 3

- 9. test deliverables

- test plan
- test design specification
- test case specification
- test procedure specification
- test item transmittal reports
- test logs
- test incident reports
- test summary reports

test conditions

test cases

steps for executing
a set of test cases

release notes

what happened

- 10. testing tasks

- including inter-task dependencies & special skills

18

Test Plan Headings 4

- 11. environment

- physical, hardware, software, tools, office space

- 12. responsibilities

- to manage, design, prepare, execute, witness, check, resolve issues, providing environment, providing the software to test

- 13. staffing and training needs

- 14. schedule

- test (and test item transmittal) milestones in project schedule

- 15. risks and contingencies

- contingency plan for each identified risk

good idea to
also include
assumptions

- 16. approvals

19

Contents

5.1 Test Organisation

5.2 Test Planning and Estimation

5.2.1 Test Planning

5.2.2 Test Planning Activities

5.2.3 Entry Criteria

5.2.4 Exit Criteria

5.2.5 Test Estimation

5.2.6 Test Strategy, Test Approach

5.3 Test Progress Monitoring and Control

5.4 Configuration Management

5.5 Risk and Testing

5.6 Incident Management

20

Test Planning Activities

- implement test policy / test strategy
- determine scope, risks and objectives of testing
- define test approach
 - levels, entry/exit criteria, test techniques, tools, etc.
- roles required, tasks to be done
- scheduling all test activities, assigning resources
- defining documentation requirements
 - what to document, templates, level of detail
- selecting metrics for monitoring and control

21

Entry Criteria

- used to define when a test activity can start
 - e.g. a test level (system test) or test activity (test execution)
- typical entry criteria:
 - test environment available and ready
 - test tool(s) available and ready
 - testable code available
 - test data available
 - testers available

22

Exit Criteria

- used to define when to stop testing
 - e.g. the end of a test level or testing goal achieved
- typical exit criteria:
 - thoroughness measures
 - ▶ such as code coverage, functionality or risk coverage
 - estimates of defect density or reliability measures
 - cost (testing budget used up)
 - residual risks
 - ▶ including defects not fixed or lack of test coverage in specific areas
 - schedules (time-based)

23

Contents

5.1 Test Organisation

5.2 Test Planning and Estimation

5.2.1 Test Planning

5.2.2 Test Planning Activities

5.2.3 Entry Criteria

5.2.4 Exit Criteria

5.2.5 Test Estimation **Проценка**

5.2.6 Test Strategy, Test Approach

5.3 Test Progress Monitoring and Control

5.4 Configuration Management

5.5 Risk and Testing

5.6 Incident Management

24

Estimating **Проценка** Testing is no Different

- estimating any job involves the following
 - identify tasks
 - how long for each task
 - who should perform the task
 - when should the task start and finish
 - what resources are available, what skills
 - predictable dependencies
 - ▶ task precedence (build test before running it)
 - ▶ technical precedence (add & display before edit)

25

Estimating Testing is Different

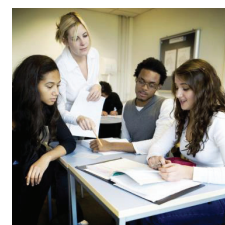
- additional destabilising dependencies
 - testing is not an independent activity
 - delivery schedules for testable items missed
 - fixed target dates
 - test environments are critical



26

Estimating Methods - Considerations

- quality of specifications
- size and complexity of application
- requirements for non-functional testing
- stability and maturity of development process
- type and location of test environments
- use of test tools
- skills of people involved
- time available
- amount of re-work required



27

Estimating Methods

- metrics-based
 - measures of previous or similar projects
 - ▶ if we have historical information or typical values
- expert-based
 - assessment by experts or task owner
 - ▶ depends on their expertise / experience
- some other approaches (for information)
 - FIA (finger in the air)
 - work breakdown structure
 - test point analysis
 - estimation model

28

Contents

- 5.1 Test Organisation
- 5.2 Test Planning and Estimation
 - 5.2.1 Test Planning
 - 5.2.2 Test Planning Activities
 - 5.2.3 Entry Criteria
 - 5.2.4 Exit Criteria
 - 5.2.5 Test Estimation
 - 5.2.6 Test Strategy, Test Approach
- 5.3 Test Progress Monitoring and Control
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

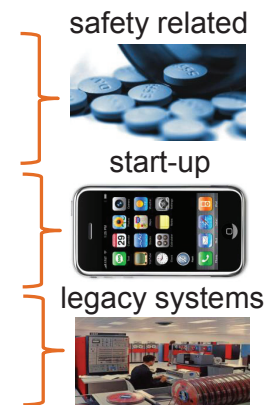
29

Test Strategies/Approaches Vary

- remember principle number 6?

6 . Testing is context dependent

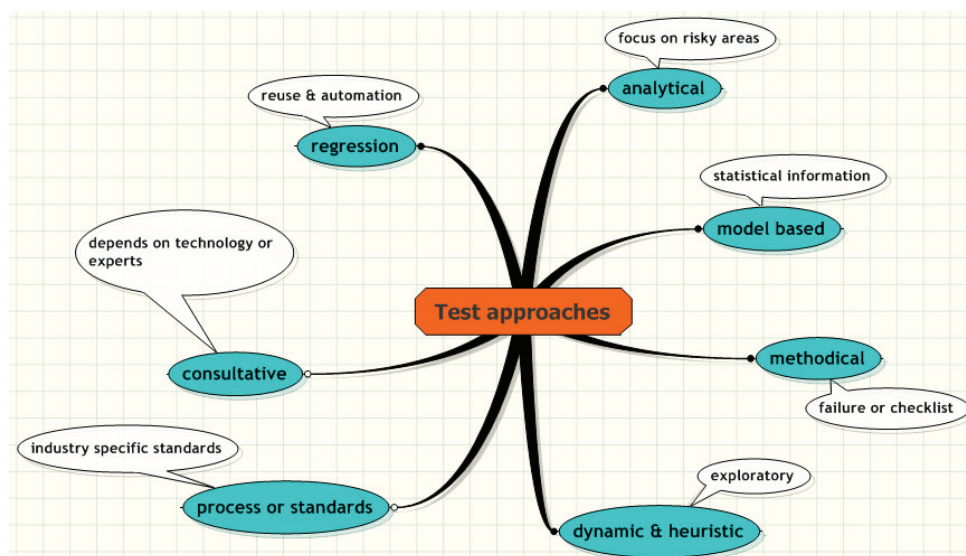
- 7 approaches determined by context
 - analytical (risk-based)
 - industry standards
 - methodical (checklist)
 - consultative (potential users)
 - dynamic/heuristic (exploratory) ← Reactive
 - regression averse (avoid regression defects)
 - model-based (operational profiles)



30

Typical Test Approaches to Consider

Different approaches can be combined !



31

Contents

- 5.1 Test Organisation
- 5.2 Test Planning and Estimation
- 5.3 Test Progress Monitoring and Control
 - 5.3.1 Test Progress Monitoring
 - 5.3.2 Test Control
 - 5.3.3 Test Reporting
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

32

Test Progress Monitoring - Reasons

- to provide feedback and visibility of testing activities
- gathering and supplying information to stakeholders so that informed decisions can be made
- show how are we doing against the plan
 - time and budget
 - exit criteria

What metrics do you keep and why?



33

Common Metrics

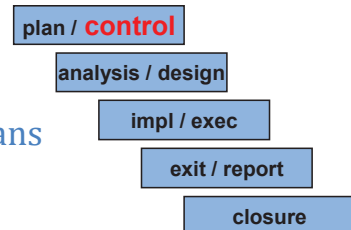
- percentages of work done
 - test cases written versus plan
 - test environment preparation
 - test execution (run/not run, passed/failed)
- progress
 - actual against milestone dates
 - costs against budget
- defect information
 - found/fixed, failure rate, defect density, retest results
- test coverage of risks, requirements or code
- subjective confidence of testers in the product



34

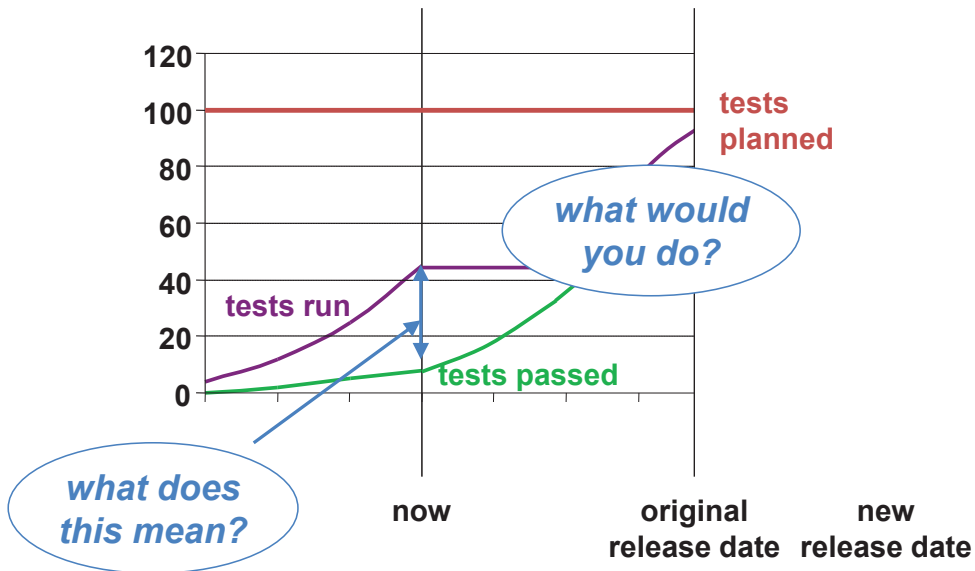
Test Control

- controlling actions are taken
 - to help us meet the original or modified plans
- some key controlling actions examples
 - more resource (people, machines, time)
 - de-scope product and / or testing
 - tighten entry criteria (better quality delivered)
 - loosen exit criteria (reduce quality to customers)
 - ...
- feedback is vital
 - this will demonstrate whether the controlling action has had the desired effect for the project.



35

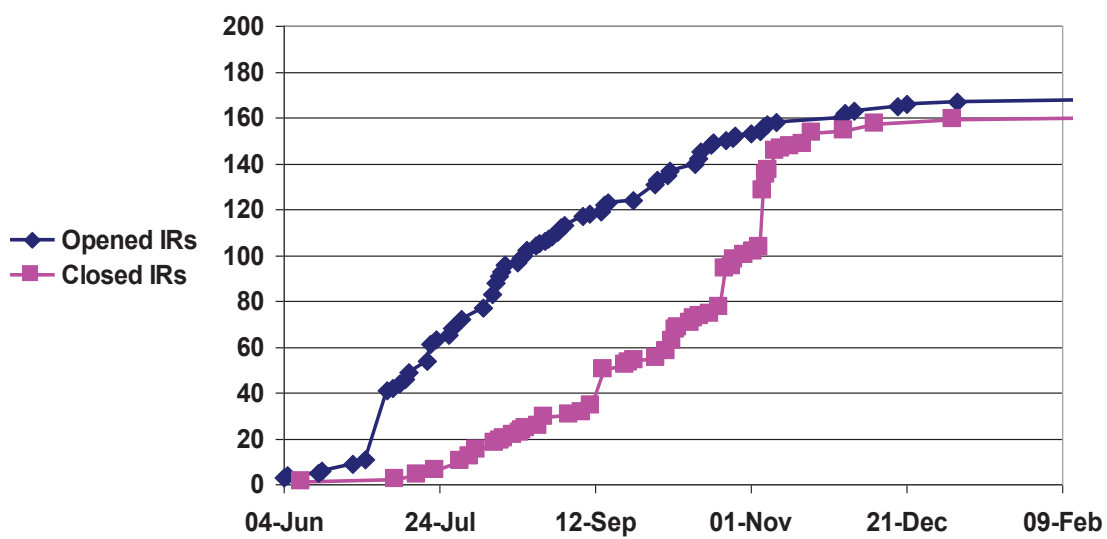
Monitoring and Controlling Test Execution Progress



36

Case History

Incident Reports (IRs)



Source: Tim Trew, Philips

37

Contents

- 5.1 Test Organisation
- 5.2 Test Planning and Estimation
- 5.3 Test Progress Monitoring and Control
 - 5.3.1 Test Progress Monitoring
 - 5.3.2 Test Control
 - 5.3.3 Test Reporting
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

38

Test Reporting

- giving a summary showing where you are with testing now
 - what happened during testing?
 - were dates and exit criteria met?
- analyse metrics and make appropriate recommendations
 - is it worth carrying on with testing?
 - what's the situation with risks?
 - are we all confident the application will work?

39

Contents of a Test Summary Report

- identifier
- summary
- variances
- comprehensive assessment
- summary of results
- evaluation
- summary of activities
- approvals

more detailed information on test summary report content is in the student notes



Source: IEEE 829 - 1998

40

Test Management Tools

- management of testware
 - test plans, specifications, results, incidents
- interface to other tools
 - test execution tools, incident logging and CM
- traceability
 - tests, test results and incidents to source documents
- reports and metrics
 - logging and reporting results,
 - quantitative analysis of tests
 - for process improvement



Special Considerations

this tool needs to interface with other tools or spreadsheets, reports need to be designed well, often required to be centrally located

41

Contents

- 5.1 Test Organisation
- 5.2 Test Planning and Estimation
- 5.3 Test Progress Monitoring and Control
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

42

Problems Resulting from Poor Configuration Management

- can't reproduce a defect reported by a customer
- can't roll back to previous subsystem
- one change overwrites another
- emergency fix needs testing but tests have been updated to new software version
- which code changes belong to which version?
- defects that were fixed re-appear
- tests worked perfectly - on old version
- "shouldn't that feature be in this version?"

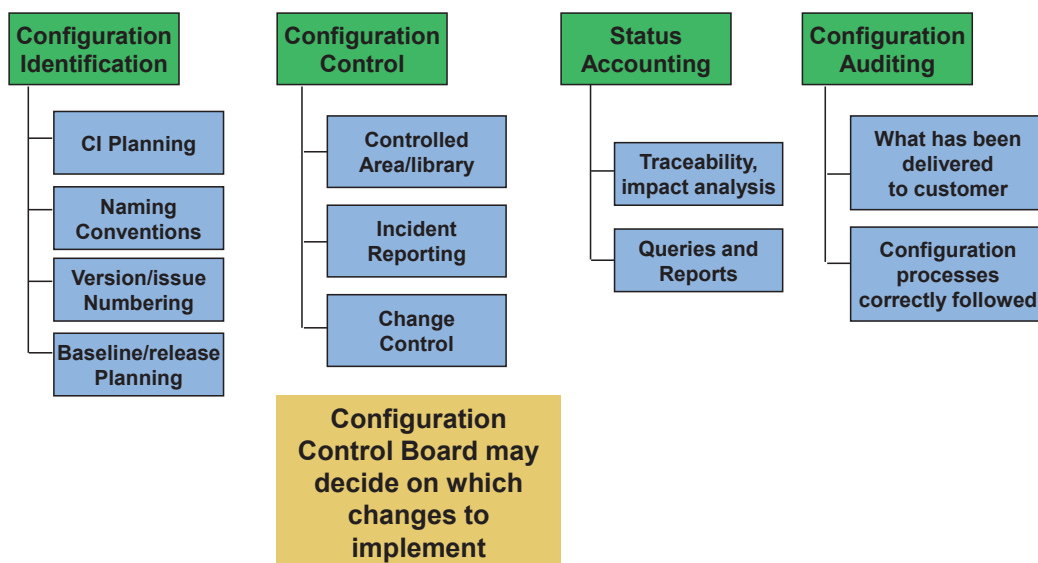
43

Configuration Management

- purpose: maintain integrity of products
 - through the product & project life cycle
- for testing:
 - all items of testware are identified, version controlled, tracked for changes
 - related to each other and related to development items (test objects)
 - traceability can be maintained throughout the test process
 - all identified documents and software items are referenced unambiguously in test documentation

44

Configuration Management Activities



45

Products for CM in Testing

- test plans
- test designs
- test cases:
 - test input
 - test data
 - test scripts / procedures
 - expected results
- actual results
- test tools

CM activities should be started during planning

CM is critical for controlled testing

What would not be under configuration management?

➡ live data!

46

Configuration Management Tools

- provide valuable infra-structure for testing to succeed
 - not strictly testing tools
- what can they do?
 - stores information about versions/builds of software
 - traceability between testware and software
 - assist with change control
 - useful when developing more than one configuration per release



47

Contents

- 5.1 Test Organisation
- 5.2 Test Planning and Estimation
- 5.3 Test Progress Monitoring and Control
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

48

Understanding Risk

- definition of risk
 - the **chance** of an event, hazard, threat or situation occurring resulting in undesirable **consequences**
- two elements
 - likelihood (probability) of a problem occurring
 - impact of the problem (if it does occur)
- impact can be on
 - product (software failure = system isn't available)
 - project (delivery timescales extended)
- risk level = likelihood * impact

49

Project Risks

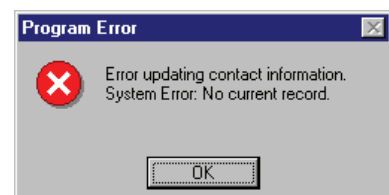
- related to the project's capability to deliver its objectives
 - for example:
 - ▶ supplier issues
 - ▶ organisational factors
 - ▶ technical issues
- these projects risks are:
 - identified as part of test planning
 - recorded in the risk register
 - addressed by project actions



50

Product Risks

- related to potential failures in the application and can impact on the resulting quality
 - for example:
 - ▶ faulty software delivered
 - ▶ potentially dangerous software
 - ▶ software does not do what it should
 - ▶ poor software characteristics (e.g. usability)
- these product risks are:
 - used to help us focus our testing effort
 - ▶ to reduce the likelihood or impact of a maturing risk
 - addressed by testing
 - ▶ may guide the types of testing chosen



51

Are These Risks? Product or Project?

- customers are abandoning your web site **No**
- software may fail to respond in time **Yes** product
- key people are already over-loaded **No**... but could lead to project risk
- computation of interest rates will be incorrect next quarter **No**
- 3rd party software may not be of expected quality **Yes** product
- 3rd party software may not be delivered on time **Yes** project
- your best tester/developer has just given notice **No** ... but could lead to project risk

52

Risk Management

- all activities needed to help reduce the chance of project and product failure
- disciplined approach to:
 - identifying risks
 - assessing risks
 - mitigating risks
 - ▶ determine which ones you can do something about
 - ▶ implement risk reduction/avoidance activities
- during testing new risks could be uncovered
- use the knowledge of experienced people to help determine the risk and mitigating activities

53

Risk-based Testing

- a proactive approach to reduce risks
 - starts during initial stages of a project
 - identify possible risks
 - clarify what problems the risks could cause
 - investigate what can be done to prevent risks from maturing
 - guides the test planning, specification, preparation and execution of tests
 - ▶ what techniques, extent of testing, test priority
 - ▶ non-testing activities (e.g. software design training)



54

Contents

- 5.1 Test Organisation
- 5.2 Test Planning and Estimation
- 5.3 Test Progress Monitoring and Control
- 5.4 Configuration Management
- 5.5 Risk and Testing
- 5.6 Incident Management

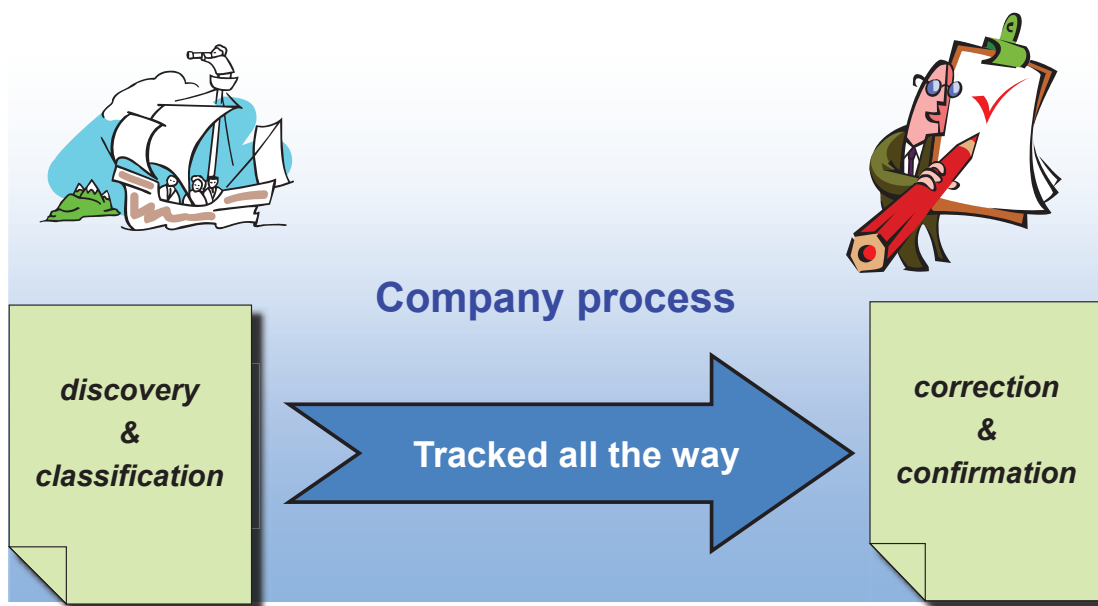
55

Incident Management

- **incident**: any unplanned event that occurs during testing that requires subsequent investigation or correction.
 - actual results do not match expected results
 - possible causes:
 - ▶ software defect
 - ▶ wrong version of the software
 - ▶ test was not performed correctly/wrong data
 - ▶ expected results incorrect
 - ▶ test environment problem
 - can be raised for documentation as well as code

56

Incident Life



57

Purpose of Incident Reports

- provide feedback
 - about a problem to assist in identification, isolation and correction
- allow test leader
 - to track quality of the system
 - to track progress
- provide ideas for test process improvement
- raised during development, review or testing
 - on code, system or documentation

58

Test Incident Report

Incident ID:	Test ID:	Software version	
Incident Description: (Include actual and expected results)		Severity:	Severity is updated during the life of the incident
		Priority:	
Name:	Date:	Attachments:	List of items being passed on to developers (e.g. screen shots)
(witnessed by: where applicable)			
Investigation results: (anomaly)		Cause Code:	Codes to record cause of defect for future root cause analysis
Name:	Date:		
Resolution:		Status:	Status of incident at any point in time
		Open : In Dev : Testing : Closed :	
Name:	Date:		
Closed By:		Date:	

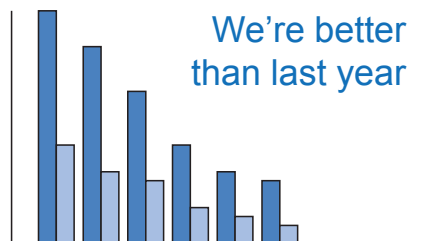
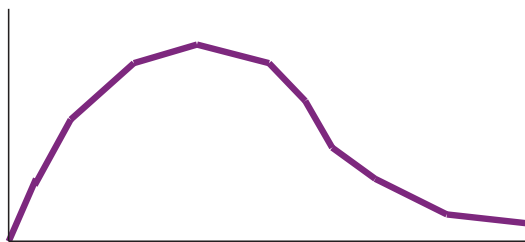
59

Severity Versus Priority

- severity
 - impact of a failure caused by this defect
- priority
 - urgency to fix a defect
- scope
 - how much of the system is affected by the problem
- examples
 - minor cosmetic typo company name
priority, not severe
 - crash if this feature is used experimental,
not needed yet:
severe, not priority

60

Use of Incident Metrics



Is this testing approach "wearing out"?

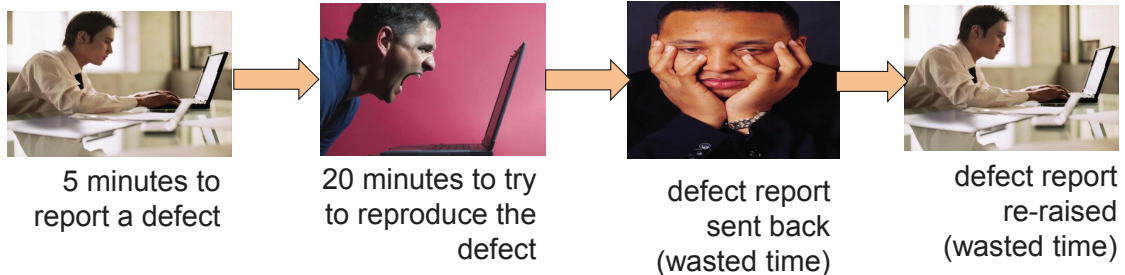


How many defects can we expect?

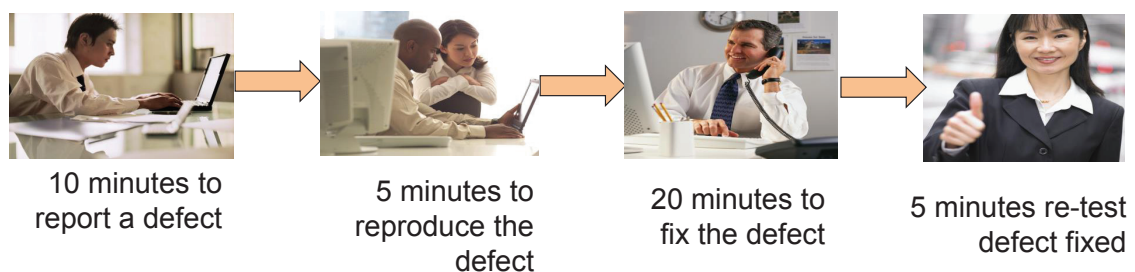
61

Report as Quickly as Possible?

scenario 1 (undesirable situation):



scenario 2 (preferred situation):



62

Incident Management Tools

- also known as defect tracking/bug logging tools
 - commercial or in-house built
- stores, manages & monitors incident reports
 - ability to log information concerning the incident (this is the tester's requirement specification)
 - ▶ priority, status, assignment, description, environment
- provides support for various analysis
 - statistical analysis & data analysis
 - reports often built in and customisable



63

Summary – Key Points

- what levels of independence: developer, buddy, tester on dev team, independent test team, business, specialist, outsource
- test documentation standard: IEEE 829
- exit criteria: coverage, risks, schedule, cost, defect density/reliability
- estimation approaches: previous metrics, experts
- approaches / strategies: analytical, model, methodical, std compliant, heuristic, consultative, regression averse
- risk: what is it, types, defined as: possible future harm, product & project, likelihood & impact

SESSION 5: TEST MANAGEMENT - NOTES

Terms

configuration management, defect density, failure rate, incident logging, incident management, incident report, product risk, project risk, risk, risk-based testing, test approach, test control, test monitoring, test strategy, test summary report, tester test leader, test manager.

From the ISTQB Glossary

configuration management: A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

defect density: The number of defects identified in a component or system divided by the size of the component or system (expressed in standard measurement terms, e.g., lines-of-code, number of classes or function points).

failure rate: The ratio of the number of failures of a given category to a given unit of measure, e.g., failures per unit of time, failures per number of transactions, failures per number of computer runs.

incident logging: Recording the details of any incident that occurred, e.g., during testing.

incident management: The process of recognizing, investigating, taking action and disposing of incidents. It involves logging incidents, classifying them and identifying the impact.

incident report: A document reporting on any event that occurred, e.g., during the testing, which requires investigation.

product risk: A risk directly related to the test object.

project risk: A risk related to management and control of the (test) project, e.g., lack of staffing, strict deadlines, changing requirements, etc.

risk: A factor that could result in future negative consequences.

risk-based testing: An approach to testing to reduce the level of product risks and inform stakeholders of their status, starting in the initial stages of a project. It involves the identification of product risks and the use of risk levels to guide the test process.

test approach: The implementation of the test strategy for a specific project. It typically includes the decisions made that follow based on the (test) project's goal and the risk assessment carried out, starting points regarding the test process, the test design techniques to be applied, exit criteria and test types to be performed.

test control: A test management task that deals with developing and applying a set of corrective actions to get a test project on track when monitoring shows a deviation from what was planned.

test monitoring: A test management task that deals with the activities related to periodically checking the status of a test project. Reports are prepared that compare the actuals to that which was planned.

test strategy: A high-level description of the test levels to be performed and the testing within those levels for an organization or programme (one or more projects).

test summary report: A document summarizing testing activities and results. It also contains an evaluation of the corresponding test items against exit criteria.

tester: A skilled professional who is involved in the testing of a component or system.

test leader: See test manager.

test manager: The person responsible for project management of testing activities and resources, and evaluation of test object. The individual who directs, controls, administers, plans and regulates the evaluation of a test object.

This section looks at the process of managing the tests and test processes. Organisational issues such as types of test teams, responsibilities, etc. are important if we wish to see effective testing within our organisation.

Strong disciplines such as Configuration Management not only assist the developer but also should be seen as a complete lifecycle discipline that manages the 'testware'.

Test management activities include risk management, estimation, monitoring, control and the recording and tracking of incidents. At any time we should know how well we are doing and understand what controlling actions we can take to keep testing on target.

During this session we also cover the typical tools that will help Test Leaders in their business, namely; Test Management, Configuration Management and Incident Management tools.

5.1 Test Organisation

Learning Objectives

LO-5.1.1	K1	Recognise the importance of independent testing.
LO-5.1.2	K2	Explain the benefits and drawbacks of independent testing within an organisation.
LO-5.1.3	K1	Recognise the different team members to be considered for the creation of a test team.
LO-5.1.4	K1	Recall the tasks of a typical test leader and tester.

Terms

tester test leader, test manager.

5.1.1 Test Organisation and Independence

The importance of independence

It is important to realise that companies will have different requirements when it comes to organisational structures for testing. The different stages of testing will be performed within organisations with varying degrees of independence using different approaches.

We have already seen that independence is important for effective testing. Greater independence gives a more objective view of the document being reviewed / inspected. Authors are less likely to take things personally and reviewers less vulnerable to pressure the more independent the testing becomes.

If we were to plot the number of failures generated over a period of time, it would probably rise steeply at first, but then begin to decline. If we were to release the product to the end users at that point, we may perhaps expect the number of failures generated to continue to diminish, but instead the number found increases. The reason for this is that the users have a different 'worldview' and now starting to use the application in live operation. Therefore if our aim is to generate as many failures as possible (and thereby discover as many defects as possible) then we need to have as many of these different worldviews as possible, and as early as possible in the lifecycle. Our challenge is to ensure that the testers find the defects instead of the users.

There are however advantages of both familiarity and independence and one should not replace familiarity with independence – we need both. Independent testers give a more objective assessment of the software and can find defects that people with a detailed knowledge of the software would not. However the programmer knows and understands the software and will know where problems are most likely to occur, so familiarity has advantages too.

Organisational structures for testing

We must recognise that while independence is important, there are varying degrees of independence which have advantages and disadvantages associated with them.

Each of the broad options is discussed in turn below. The detail of each is unlikely to be required in the exam.

For large, complex or safety critical projects it is appropriate to have different levels of testing performed by independent testers. One could bring in developers to be involved in testing, particularly at the lower levels, but often their effectiveness is limited as they may not be as objective as an independent tester.

Developers Only

This is where the programmer will test his or her own code. They know the code best and are more familiar with it, so they may find problems that a less technical tester would miss. They can also find and fix defects very cheaply at this stage.

However, they might not be the best people to try to break it. There is a tendency to see what you meant instead of what is actually there, so they may miss things that an independent mind would see. It may also be a rather subjective assessment of their own work; they want to show how good they are, not how easily their software can be made to fall over! There is also the danger their interpretation of the requirements may be wrong, yet it is used to build and then test the code.

Developers test each other's code

This is where a developer will test another developer's code, and vice versa. This can be less threatening than having an independent tester, especially if developers know each other and understand how the other will code. Another developer may better understand their strengths and weaknesses and have an idea of where problems may be.

The downside though maybe due to their own pressure of development work, not enough attention may be given to looking at the other person's code.

Tester(s) on the Development Team

One of the members of a development team is assigned the responsibility for testing. This person may already be an experienced tester (with or without development experience) and could be brought into the team specifically to take the testing responsibility. Although working along-side the developers, this person may not have a detailed knowledge of the system from a technical perspective. This gives greater independence in their testing and yet encourages a team spirit in which developers and tester are working toward the same goal.

However, testers in this situation might find themselves undermined and unreasonable pressure placed on them to do all the testing because it is deemed to be 'their job'. They may be corruptible by peer pressure and where it is only one tester on the team it provides only a single view.

Independent test team or group

This team may be referred to as an Independent Test Group (ITG) or Independent Test Unit (ITU) and will usually be totally independent of development with a different reporting structure. These teams usually are looked upon as the 'testing experts' and will have a high level of testing experience.

There can, however, be a high degree of over-reliance on this team to perform all necessary tests including those that should be undertaken by developers (particularly component testing). Alternatively little or no component testing may be performed, leaving the independent testers to find coding defects that could have been found and fixed more cheaply had component testing been done. In these situations the independent testing team becomes a bottleneck and test responsibility, rather than shared, is left to the ITG.

A completely separate department has its advantages, but it does also have drawbacks – namely confrontation and an 'over-the-wall' mentality. Both of these issues need to be resolved if we are to see an effective and efficient test regime.

Independent testers from business/user community

It would always be a good idea to involve people to do some testing from the business/user areas. These are the only people who may have a real grasp of what is expected to be required from the system and will test it with that perspective. They could also be the only group of people with the knowledge and skill to understand where the system fits into the wider scheme of applications or processes. Their knowledge is therefore very valuable.

However, sometimes testing to them is seen as a lower priority to their other work and they may not give the time it deserves or is expected. This can lead to frustration and conflict with

other project members unless carefully managed. We can also find that less knowledgeable people are chosen for the testing role and therefore the necessary experience is lacking.

Independent test specialists

There may be situations where specialist skills are needed for specific types of testing. Examples can be security, performance and usability etc. These are people who have chosen to focus and specialise in one discipline of testing. Because testing has such wide ranging skills, it is not ever possible for one person to know everything. That is when specialist people can help. Such specialist people may come from an outside organisation, given their narrow focus areas.

Such people can be in demand in larger organisations to work on different projects and applications. As such they may not have the business knowledge for any one application. As they may have chosen a specific career route in a specialist discipline they may not be too keen to help out on the more general testing activities. Often their specialist skills are in great demand (particularly security and performance testers) and as such are liable to be more expensive.

Outside Organisation

Some companies provide a testing service. This can be undertaken on their site or they may send a number of testers to manage and perform the testing on the developer site. These companies usually specialise in a certain industry (such as insurance, finance, banking, etc.) and so can provide in depth specialist business knowledge. There are also organisations that will focus on specialist areas e.g. security and performance.

As they are outside organisations, it is unlikely they will be drawn into internal politics. They can sometimes be considered expensive and any experience they gain will be lost from the project once their testing is complete. However, if faced with such questions you always need to consider what the alternatives could be. If this solution allows you to meet a critical release date and gain market share, the cost may not be significant compared to the business benefits.

Overall benefits

Overall benefits and pitfalls can be condensed into:

Benefits:

- Independent testers see other and different defects and are generally unbiased.
- An independent tester can verify assumptions people made during specification and implementation of a system.

Drawbacks:

- Can be too isolated from development team.
- Independent testers may be perceived to be a bottle neck and/or blamed for delays to the software release.
- Developers may lose sense of responsibility for quality and feel others will pick it up.

As an example, some possible levels of independence at each of the testing levels could be:

- component testing - performed by programmers;
- system testing - performed by an independent test team;
- acceptance testing – performed by users or user representatives;
- operational acceptance testing – performed by operational representatives.

Resource issues

This section provides additional information beyond that required by the syllabus.

- Independence is important in testing, as an independent mind will see things that may be missed by the person who developed the software. However, familiarity also has benefits. It is not a question of achieving one or the other, but of achieving a good

balance. Different levels of testing can use different approaches to achieve independence. For example, the use of test design techniques gives independence of thought. A test strategy should state what levels of independence are required for each level of testing.

- A good mix of skills is important within a project. We must however consider the skill set for the team. The following are a useful reminder of the sort of skill set we will need in order to facilitate good testing.
- Technique Specialists – specialise in the use of test design techniques such as Equivalence Partitioning and Boundary Value Analysis. They can then become the source of knowledge, advice and guidance for the rest of the team;
- Automation Specialists – These have a keen understanding and desire to specialise in the test automation arena. They can develop automation standards and promote good automation practices. They generally need programming skills, since test scripts for automated tools are programming languages;
- Database Experts – Everyone in the team should have an understanding of the underlying database, but not everyone needs know the intricate details of the database environment.
- Business Skills – Having key people in the test team with business knowledge is essential for testing from a business perspective;
- Usability Experts – We have already seen that testing for usability is often poorly done because it is often poorly specified. Test (and development) teams could benefit by having specialised expertise in this area;
- Test Environment Experts – Maintaining test environments is crucial for successful testing and this task should not be underestimated in terms of its complexity and sensitivity;
- Test Leaders – Can be also know as Test Managers. Essentially those people, who encourage, motivate and protect the rest of the test team.
- Communicator – One of the key things in testing is communication. This is because you will be speaking different things to different people i.e. developers, business, marketing, customer etc. You will need to be able to effectively know and translate between areas.

5.1.2 Tasks of the Test Leader and Tester

The syllabus divides testing tasks into those typically done by test leaders (which according to the syllabus is the same as a test manager), and testers.

Test Leader tasks

Typical tasks of the test leader include those to do with planning, monitoring and controlling the testing. Here is the more complete (though not exhaustive) list taken from the syllabus.

- Coordinate the test strategy and plan with project managers and others.
- Write or review a test strategy for the project, and test policy for the organization.
- Contribute the testing perspective to other project activities, such as integration planning.
- Plan the tests – considering the context and understanding the risks – including selecting test approaches, estimating the time, effort and cost of testing, acquiring resources, defining test levels, cycles, approach, and objectives, and planning incident management.
- Initiate the specification, preparation, implementation and execution of tests, and monitor and control the execution.
- Adapt planning based on test results and progress (sometimes documented in status reports) and take any action necessary to compensate for problems.
- Set up adequate configuration management of testware for traceability.
- Introduce suitable metrics for measuring test progress and evaluating the quality of the testing and the product.
- Decide what should be automated, to what degree, and how.
- Select tools to support testing and organize any training in tool use for testers.

- Decide about the implementation of the test environment.
- Schedule tests.
- Write test summary reports based on the information gathered during testing.

Tester tasks

Typical tester tasks include tasks to do with actually carrying out the test design and execution, raising incidents, etc. Here again is the more complete (though not exhaustive) list taken from the syllabus.

- Review and contribute to test plans.
- Analyse, review and assess user requirements, specifications and models for testability.
- Create test conditions, test cases and test procedures.
- Set up the test environment (often coordinating with system administration and network management).
- Prepare and acquire test data.
- Implement tests on all test levels, execute and log the tests, evaluate the results and document the deviations from expected results.
- Automate tests (may be supported by a developer or a test automation expert).
- Test non-functional characteristics such as performance.
- Review tests written by the team.

Test Leaders/Managers and testers have a range of responsibilities to effectively deliver projects. As a result the roles demand that you have a lot of knowledge about a large number of areas rather than specialising in a narrow band. In a practical way, some of these tasks could easily be included in job descriptions!

In our experience the roles are split so that clear responsibilities are understood. The split of the roles should be clearly shown in the test strategy so everyone is aware what is expected of them. Detailed work roles can be defined in the Test Plan.

Who does testing?

The testing levels undertaken can have many different people involved. Examples of the typical roles are:

- Test leaders and testers – performing the planning, estimation and execution etc.
- However, there are situations where testing tasks could also be filled by other roles within a project as follows:
- project manager – managing whole project to ensure delivery milestones are adhered to. In a small company, the project manager may also actually do some of the testing;
- quality manager – looking to see if applications meet internal company standards and making sure the look and feel is right;
- developer – white box testing and providing technical support to other levels of testing;
- business/domain expert – ensuring the product meets user expectations and/or planned requirements;
- infrastructure – a database administrator or system administrator may be involved in testing those aspects of the system that they will be responsible for, or they may be involved in integration testing with other systems.

5.2 Test Planning and Estimation

Learning Objectives

- | | | |
|----------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LO-5.2.1 | K1 | Recognise the different levels and objectives of test planning. |
| LO-5.2.2 | K2 | Summarise the purpose and content of the test plan, test design specification and test procedure documents according to the 'Standard for Software Test Documentation' (IEEE Std 829-1998). |
-

LO-5.2.3	K2	Differentiate between conceptually different test approaches, such as analytical, model-based, methodical, process/standard compliant, dynamic/heuristic, consultative and regression-averse.
LO-5.2.4	K2	Differentiate between the subject of test planning for a system and scheduling test execution.
LO-5.2.5	K3	Write a test execution schedule for a given set of test cases, considering prioritization, and technical and logical dependencies.
LO-5.2.6	K2	List test preparation and execution activities that should be considered during test planning.
LO-5.2.7	K1	Recall typical factors that influence the effort related to testing.
LO-5.2.8	K2	Differentiate between two conceptually different estimation approaches: the metrics-based approach and the expert-based approach.
LO-5.2.9	K2	Recognise/justify adequate entry and exit criteria for specific test levels and groups of test cases (e.g., for integration testing, acceptance testing or test cases for usability testing).

Terms

test approach, test strategy.

5.2.1 Test Planning

The test planning process and scope of testing to be performed is influenced by a number of areas notably:

- the test policy;
- objectives of testing;
- the risks to the project/product;
- any constraints identified;
- the criticality of the application;
- the availability of resources (not just people but also networks, environments, etc.).

All these elements feed into the different test plan documents that can be issued. However, when you start planning the testing, you will not know all the answers, but that should not stop you in producing a first cut of the documents. As time passes and more information is gleaned you should update the appropriate test plans.

5.2.2.1 Project/Master test plan

The purpose of high level test planning is to produce a high-level test plan! A high-level test plan is synonymous with a project test plan and covers all levels of testing. It is a management document describing the scope of the testing effort, resources required, schedules, etc.

There is a standard for test documentation. It is ANSI/IEEE 829 "Standard for Software Test Documentation". This outlines a whole range of test documents including a test plan. It describes the information that should be considered for inclusion in a test plan using 16 key headings.

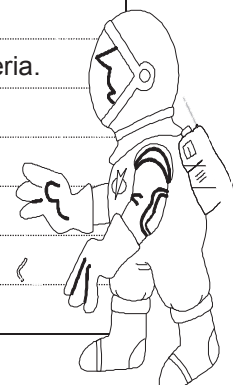
Given this is shown as a K2 level you only need to know the purpose and contents of the test plan according to IEEE 829 - 1998. Details of the headings are shown below for further information.

1. Test Plan Identifier: A unique reference for this document.
2. Introduction: A guide to what the test plan covers and references to other relevant documents such as the Quality Assurance and Configuration Management plans.
3. Test Items: The physical things that are to be tested such as executable programs, data files or databases. The version numbers of these, details of how they will be handed over to testing (on CD, wiki, intranet, etc.) and references to relevant documentation.

4. **Features to be Tested:** The logical things that are to be tested, i.e. the functionality and features.
5. **Features not to be Tested:** The logical things (functionality / features) that are not to be tested.
6. **Approach:** The activities necessary to carry out the testing in sufficient detail to allow the overall effort to be estimated. The techniques and tools that are to be used and the completion criteria (such as coverage measures) and constraints such as environment restrictions and staff availability.
7. **Item Pass / Fail Criteria:** For each test item the criteria for passing (or failing) that item such as the number of known (and predicted) outstanding defects. We need to know when we stop testing or move onto a new phase of testing. There are a number of measurable criteria we can identify, document and then follow them during the project. It is important that whatever measure is used it is practical to do so. All persons involved also need to be aware and agree the measures to be put in place. The test plan will document the criteria to be used and by gaining sign off from the relevant people to the plan, they are committing to meet the set criteria.
8. **Suspension / Resumption Criteria:** The criteria that will be used to determine when (if) any testing activities should be suspended and resumed. For example, if too many defects are found with the first few test cases it may be more cost effective to stop testing at the current level and wait for the defects to be fixed.
9. **Test Deliverables:** What the testing processes should provide in terms of documents, reports, etc.
10. **Testing Tasks:** Specific tasks, special skills required and the inter-dependencies.
11. **Environment:** Details of the hardware and software that will be needed in order to execute the tests. Any other facilities (including office space and desks) that may be required.
12. **Responsibilities:** Who is responsible for which activities and deliverables?
13. **Staffing and Training Needs:** Staff required and any training they will need such as training on the system to be tested (so they can understand how to use it) training in the business or training in testing techniques or tools.
14. **Schedule:** Milestones for delivery of software into testing, availability of the environment and test deliverables.
15. **Risks and Contingencies:** What could go wrong and what will be done about it to minimise adverse impacts if anything does go wrong.
16. **Approvals:** Names and when approved.

To help you remember what is and is not included in a test plan, consider the following table that maps all of the headings onto the acronym SPACE DIRT.

S cope	Test Items, Features to be Tested, and Features not to be Tested.
P eople	Staffing and Training Needs, Schedule, Responsibilities.
A pproach	Approach.
C riteria	Item Pass/Fail Criteria, Suspension and Resumption Criteria.
E nvironment	Environment.
D eliverables	Deliverables
I ncidental	Test Plan Identifier, Introduction, Approvals
R isks	Risks and Contingencies
T asks	Testing Tasks



5.2.2 Test Planning Activities

Test planning activities for an entire system or part of a system may include the following:

- implementing the test policy and/or strategy that is applicable;
- determining the scope, risks and objectives of testing;
- defining the overall test approach (techniques, what is to be tested (test items), coverage, testware, test levels, entry and exit criteria;
- integrating test activities in the software development life cycle;
- deciding what to test, which roles will undertake each of the testing tasks;
- determining test resources (people, hardware, software, environment);
- schedule test tasks (analysis & design of tests, implementation and execution of test procedures, etc.);
- defining test documentation requirements (which documents, how formal);
- selecting metrics for monitoring and control of test activities.

The test planning activity produces a test plan specific to a level of testing (e.g. system testing). These test level specific test plans should state how the test strategy and project test plan apply to that level of testing and state any exceptions to them. When producing a test plan, clearly define the scope of the testing and state all the assumptions being made. Identify any other software required before testing can commence (e.g. stubs & drivers, test tools or other 3rd party software) and state the completion criteria to be used to determine when this level of testing is complete.

5.2.3 Entry Criteria

The purpose of entry criteria is to define under what conditions it will be acceptable to start testing. They can be applied to the beginning of a test level or to a specific set of tests.

Typical entry criteria are (using a combination of criteria is usually better than using just one):

- test environment availability and readiness;
- test tool readiness in the test environment;
- testable code availability;
- test data availability;
- testing at the previous level has been completed.

5.2.4 Exit Criteria

The purpose of exit criteria is to define the conditions under which it will be acceptable to stop testing. Exit criteria typically apply to a specific level of testing (such as system or acceptance testing) but may also be used for other test efforts such as a group of tests with a specific objective such as performance or usability testing.

Example exit criteria are (some are better than others and using a combination of criteria is usually better than using just one):

- thoroughness measures such as code coverage, functionality or risk coverage
- estimates of defect density or reliability measures
- cost (testing budget used up)
- residual risks (including defects not fixed or lack of test coverage in specific areas)
- schedules (time-based)

Note that the exit criteria from one stage of testing may form the entry criteria to the next stage, but there may be others not directly related. For example an entry criterion to System Test may be that the test environment has been set up, but this is an environment only for system testing, not used in earlier stages of testing.

5.2.5 Test Estimation

Estimating testing is no different

Test estimation in many ways is no different to estimating other activities such as software design or programming. We must break down the activities into well defined tasks that, in turn, we have a better chance of estimating how long it would take to do that piece of work.

Estimating testing is different

Whilst test estimation is similar to estimating any other activity in some respects, in others ways it is very different. The main reasons are below. Testing is:

- not an independent activity – testing is completely reliant on development delivering the software to an agreed date and to an agreed quality standard. Should the quality of the software not be as good as we expected then we will spend more time reporting an greater number of defects and retesting them;
- reliant on attaining the agreed system – that there are no new surprise features added by the developers. If extra features are added, then these will need testing and this will affect the schedules;
- reliant on a stable test environment – if the test environment is volatile then this again will affect the schedules.

Estimating considerations

When undertaking the estimation, you need to be mindful of a number of different elements. The sorts of things you need to be aware of and take into account are listed below. Some or all may apply in each case.

- the quality of the specifications
- the size and complexity of the application being testing
- requirements for non-functional testing
- the stability and maturity of the development process used
- the type and location of test environments
- the use of tools to support testing
- the skills of the people involved
- the time available
- the amount of re-work required (which depends on the number of defects injected before tests are executed)

It is very important that rework is planned for. If you don't plan for re-work, then you are effectively saying one of three things:

- there won't be any defects
- if there are any defects, we won't fix them
- if we fix them, we don't care if they are fixed correctly or if they have any effect elsewhere in the system

Estimating methods

There are a number of methods that can be employed to estimate the testing effort required, which are described below. However the syllabus concentrates on just two which are:

- past project knowledge. To base our estimates on previous, similar projects is a reasonable thing to do. This is only effective if we have recorded such data from previous projects. Again, it can be easily challenged and estimates may be reduced as a result;
- an assessment of experts for the amount of effort that will be required. When you don't have any prior knowledge of the application to base your estimate on, then you can seek the views of a few "experts". Ask how long they consider the testing should take and then add up all the individual estimates and divide by the number of people asked, to get an average. If two have widely different figures, ask them to agree

between them what that figure should be and advise you. This technique whilst sounding a bit like guess work is based on the scientific approach known as “wisdom of crowds”.

The other techniques you to consider are:

- Guessing (Finger in the Air – F.I.A. approach). This is not a good method to base our final estimate on, because it can be easily questioned and very often challenged. Whilst it is not advisable to rely solely on this method, it does have its use and can be reasonably accurate depending on past project experience and the expertise of the estimator.
- Work Breakdown Structures (WBS). Here we identify tasks that make up the test activities and estimate each one in turn. Testers who could verify that the estimate is realistic can then review each task estimate. If not then estimates would be re-worked. Should the estimates not be approved, then each task in turn can be questioned as to relevance, criticality, urgency and importance – the estimates can then be adjusted accordingly.
- Fixed budget – where we are allowed a fixed amount of funds and we need to do what testing we can achieve within the constraints. We need to do a good risk assessment so we know what the impact can be if certain tests cannot be performed.
- Test point analysis – developed by Martin Pol et al as part of TMap®. Defined as a method to measure test size of an information system on the basis of a function point analysis but considering the risk element.

Whatever method is used, test estimation will always be required in advance and should be included in the high level test plan. It is recommended that you also use two methods to compare one with the other. The estimates should be reviewed and if necessary revised throughout the project once further information is obtained.

Estimating iterations

There is one major difference in estimating testing compared to estimating other tasks. Most activities, once they are done, that's it, and they are finished and complete. However, testing, once it is done it does not stay "done" - it has to be done again and again. Successful tests will find defects, but once they are fixed, re-tests and regression tests are needed. This can result in a number of test iterations or test cycles. Three or four test iterations are typical. It may not be necessary to perform all of the tests with every iteration (often it is not possible because of time constraints) but it is certainly desirable to do so with the last iteration (though this too is more often an ideal than a reality).

Past history is often a good guide to the likely number of iterations. For example, if the release of the system underwent five test iterations then there is a good chance that the next release will need at least four and possibly six.

An added complication to estimating iterations is that not all iterations will use all of the tests. Some may contain only checks for correct fixes, for example, while others may be a complete regression test of all tests in a suite.

5.2.6 Test Strategy, Test Approach

The syllabus considers that “test approach” and “test strategy” are the same thing.

We can classify general test approaches by the time at which the main bulk of test design work is done:

- **Preventative:** deciding on the approaches to take as soon as possible to ensure we consider all the elements we know about (i.e. do the test design as early as possible)
- **Reactive:** responding to the application when we actually see it or have access to it (i.e. test design done later).

Generally proactive approaches are better at least for some of the testing.

The approaches we take vary depending on the context of the application and may consider risks, available resources and skills, the technology, the nature of the system (e.g., custom

built vs. COTS), test objectives, and regulations. The following are the approaches as listed in the syllabus.

- Analytical approaches, such as risk-based testing where testing is directed to areas of greatest risk.
- Model-based approaches, such as testing using statistical information about failure rates (such as reliability growth models) or usage (such as operational profiles).
- Methodical approaches, such as failure based (including error guessing and fault-attacks), check-list based, and quality characteristic based.
- Process- or standard-compliant approaches, such as those specified by industry-specific standards or the various agile methodologies.
- Dynamic and heuristic approaches, such as exploratory testing where testing is more reactive to events than pre-planned, and where execution and evaluation are concurrent tasks.
- Consultative approaches, such as those where test coverage is driven primarily by the advice and guidance of technology and/or business domain experts outside the test team.
- Regression-averse approaches, such as those that include reuse of existing test material, extensive automation of functional regression tests, and standard test suites.

It is possible to combine approaches such as a dynamic risk-based approach, or a consultative model-based approach.

The best test approach to use depends on the context of your organisation. Consider the following factors when choosing an approach / strategy:

- risks (project and product risks – see later section);
- people involved, their skills and experience;
- objective or mission of the testing;
- regulatory requirements;
- the company's product and/or core business.

5.3 Test Progress Monitoring and Control

Learning Objectives:

LO-5.3.1	K1	Recall common metrics used for monitoring test preparation and execution.
LO-5.3.2	K2	Explain and compare test metrics for test reporting and test control (e.g., defects found and fixed, and tests passed and failed) related to purpose and use.
LO-5.3.3	K2	Summarise the purpose and content of the test summary report document according to the 'Standard for Software Test Documentation' (IEEE Std. 829-1998).

Terms

defect density, failure rate, test control, test monitoring, test summary report.

5.3.1 Test Progress Monitoring

Once we have started testing we must monitor our test progress and then take appropriate corrective action should things go wrong. Effective monitoring and control is vital in the test management process. Monitoring and control go together very well. It can be one thing to monitor testing but within that activity you also need the control element i.e. monitoring will show us moving away from the plan, whereas control is saying what can we do to bring ourselves back on track again.

As testers we need to know where we are at any point in time against the plan and be able to report on it. There can be a wealth of information available and it is our task to use a sensible amount and make it meaningful.

Recording the number of tests run against the number of tests passed and the number of tests planned is one good way to show test progress. This is a powerful visual aid when plotted on a graph. For example, if the number of tests passing falls significantly below the number run there may be a number of different reasons. Perhaps there are lots of defects being found or there are only one or two defects that are affecting many tests. In either case more development effort needs to be resourced to fix the defects. A faulty test environment could cause the problem, so in this case more development resource may be needed to resolve it.

Other useful measures include the number of incidents or defects raised together with either the number resolved and / or the number of defects expected.

5.3.2 Test Control

Test control is about management actions and decisions that affect the testing process, tasks and people with a view to making a testing effort achieve its objectives. This may be the original or a modified plan. Modifying an original plan in the light of new knowledge (i.e. what testing has revealed so far) is a frequently necessary and prudent step.

The use of entry and exit criteria are perhaps one of the simplest and yet highly effective control mechanisms available to managers. Entry criteria are conditions that must be met before the associated activity can start. Similarly, exit criteria are conditions that must be met before an activity can be declared complete.

The exit criteria of one activity are often the same as the entry criteria of the next activity. For example, the exit criteria for component testing might be that all components have been tested sufficiently to achieve 100% statement coverage and all known defects have been fixed. These could also be the entry criteria for the next testing activity (supposedly integration in the small, but could also apply to system testing). However, the next activity might have additional entry criteria. For example, entry criteria for system testing might include something about the availability of the test environment.

Tightening (or loosening) an entry or exit criteria is just one of the actions a test manager can take. Reallocation of resources such as acquiring more testers or developers, and moving people from one task to another to focus attention on more important areas is often an effective controlling action.

Other factors are re-prioritising tests when a risk occurs or matures (e.g. software is delivered late or is of poor quality), changing the test schedule due to availability or not of the test environment, setting the exit criteria for developers when fixes have been retested.

There are some factors that testing can affect indirectly, such as which defects are fixed first. However, one thing that cannot be affected by testing is the number of defects that are already in the software being tested. The testing only affects whether or not those defects are found.

Once a controlling action has been taken some form of feedback is essential in order to see the effect of the action.

Neither the testers nor the test manager should make the decision about when to release a product. The testers and test manager are responsible for supplying accurate and objective information about the software quality so that whoever does make the release decision makes it on the basis of solid facts.

5.3.3 Test Reporting

There are a vast number of metrics you can monitor during testing. It is important to only select and use those that will tell you something and give value to the project without duplicating effort and information. The main areas to consider are tests that are built then run against the plan, how many are passing/failing, how you are doing against milestones in the plan and costs against budget. Any variances need to be fully explained so it is not left to

individual interpretation of the results. Details of a test summary report are found in IEEE 829-1998.

However, it is important to collect metrics constantly during the different test levels to help you understand:

- the adequacy of the test objectives for that level of testing;
- the adequacy of the test approaches taken;
- the effectiveness of the testing with respect to its objectives.

5.3.4 Test Management Tools

These are covered fully under section 6. However there are specific tools to help in test management. These have the ability to help with:

- management of testware (anything you need to do the testing);
- an interface to other tools you may already have in the organization;
- traceability of tests to requirements and also the other way around;
- producing progress/status reports and metrics.

5.4 Configuration Management

Learning Objectives:

LO-5.4.1 K2 Summarise how configuration management supports testing.

Terms

configuration management.

What is configuration management?

Our systems are made up of a number of items (or things). Configuration Management is all about effective and efficient management and control of these items.

During the lifetime of the system many of the items will change. They will change for a number of reasons; new features, defect fixes, environment changes, etc.

An indication of a good Configuration Management system is to ask ourselves whether we can go back two releases of our software and perform some specific tests with relative ease.

Problems resulting from poor configuration management

Often organisations do not appreciate the need for good configuration management until they experience one or more of the problems that can occur without it. Some problems that commonly occur as a result of poor configuration management systems include:

- the inability to reproduce a defect reported by a customer;
- two programmers have the same module out for update and one overwrites the other's change;
- unable to match object code with source code;
- do not know which fixes belong to which versions of the software;
- defects that have been fixed reappear in a later release;
- a defect fix to an old version needs testing urgently, but tests have been updated.

Definition of configuration management

A good definition of configuration management is given in the ANSI/IEEE Standard 729-1983, Software Engineering Terminology. This says that configuration management is:

- “the process of identifying and defining Configuration Items in a system,
- controlling the release and change of these items throughout the system life cycle,

- recording and reporting the status of configuration items and change requests, and
- verifying the completeness and correctness of configuration items.”

This definition neatly breaks down configuration management into four key areas:

- configuration identification;
- configuration control;
- configuration status accounting; and
- configuration audit.

Configuration identification is the process of identifying and defining Configuration Items in a system. Configuration Items are those items that have their own version number such that when an item is changed, a new version is created with a different version number. So configuration identification is about identifying what are to be the configuration items in a system, how these will be structured (where they will be stored in relation to each other) the version numbering system, selection criteria, naming conventions, and baselines. A baseline is a set of different configuration items (one version of each) that has a version number itself. Thus, if program X comprises modules A and B, we could define a baseline for version 1.1 of program X that comprises version 1.1 of module A and version 1.1 of module B. If module B changes, a new version (say 1.2) of module B is created. We may then have a new version of program X, say baseline 2.0 that comprises version 1.1 of module A and version 1.2 of module B.

Configuration control is about the provision and management of a controlled library containing all the configuration items. This will govern how new and updated configuration items can be submitted into and copied out of the library. Configuration control also determines how defect reporting and change control is handled (since defect fixes usually involve new versions of configuration items being created).

Status accounting enables traceability and impact analysis. A database holds all the information relating to the current and past states of all configuration items. For example, this would be able to tell us which configuration items are being updated, who has them and for what purpose.

Configuration auditing is the process of ensuring that all configuration management procedures have been followed and of verifying the current state of any and all configuration items is as it is supposed to be. We should be able to ensure that a delivered system is a complete system (i.e. all necessary configuration items have been included and extraneous items have not been included).

Configuration management in testing

Just about everything used in testing can reasonably be placed under the control of a configuration management system. That is not to say that everything should. For example, actual test results may not be though in some industries (e.g. pharmaceutical) it can be a legal requirement to do so.

The process of Configuration management needs to be instigated as part of the planning process and as a result will ensure all testware items are identified, changes tracked, there is traceability during the test process and all relevant documents are referenced in test documentation.

Note that configuration management can be done using specialised configuration management tools or can be done manually.

Configuration management tools

Whilst they are not strictly testing tools, they are essential to help in each phase of testing. They are specifically designed to make the process of configuration management as easy as possible. They will:

- store information about versions of software;
- provide traceability between testware and software;
- assist in change control;
- help when developing more than one configuration per release.

5.5 Risk and Testing

Learning Objectives:

LO-5.5.1	K2	Describe a risk as a possible problem that would threaten the achievement of one or more stakeholders' project objectives.
LO-5.5.2	K1	Remember that the level of risk is determined by likelihood (of happening) and impact (harm resulting if it does happen).
LO-5.5.3	K2	Distinguish between the project and product risks.
LO-5.5.4	K1	Recognise typical project and product risks.
LO-5.5.5	K2	Describe, using examples, how risk analysis and risk management may be used for test planning.

Terms

product risk, project risk, risk, risk-based testing.

Definition of risk

A risk is an event that may or may not happen. If it has already happened, then it is no longer a risk but becomes an issue in that something needs to be done. If the likelihood of it happening is 100%, then it is a (future) certainty. One element of a risk is the likelihood or probability that the event will actually occur. For example, there is a risk of rain tomorrow, but the sun coming up is not a risk.

The other aspect to risk is the impact of the event if it does occur. Some events have very severe consequences; others have very minor effects. For example, if I am stung by a bee, it will probably only be a minor irritation, but if I were allergic to bee stings, it could kill me. So the risk of a bee sting is significantly higher for someone who is allergic.

Combining these two factors gives us a single measure of risk called "risk level":

Risk level = likelihood x impact

5.5.1 Project Risks

A project risk is to do with the project's capability to deliver its objectives and includes the considerations outlined below.

- Organisational factors such as:
 - skill, training and staff shortages;
 - personnel issues;
 - political issues such as problems with testers communicating their needs and test results objectively and clearly, failure by the team to improve development and testing practices over time;
 - improper attitude toward or expectations of testing (e.g. not appreciating the value of finding defects during testing)
- Technical issues such as:
 - problems in defining the right requirements;
 - the extent to which requirements cannot be met given existing constraints;
 - test environment not ready on time;
 - late data conversion, migration planning and development and testing data conversion/migration tools;
 - low quality of the design, code, configuration data, test data and tests.

- Supplier issues such as:
 - failure of a third party;
 - contractual issues.

5.5.2 Product Risks

Some risks apply to products. For example there may be a risk that your software doesn't interact with an obscure browser or application - this is a product risk. Product risks are to do with the potential failure areas in the software of system as they have the potential to impact the quality of the product. They include the following considerations:

- failure-prone software delivered
- the potential that the software/hardware could cause harm to an individual or company
- poor software characteristics (e.g., functionality, reliability, usability and performance)
- poor data integrity and quality (e.g., data migration issues, data conversion problems, data transport problems, violation of data standards)
- software that does not perform its intended functions

Risk based approach

We are looking at a risk based approach or using risk analysis to direct where the testing should concentrate, and to prioritise the testing for the product. Project risks can be used to be a proactive approach to reduce the probability of risks maturing or reduce the potential impact. It also helps when making a release decision to understand what the chance of failures could be. For example if the testing planned has not been carried out completely due to time pressure, there is an increased risk of failure in operation. This approach is started as early as possible to identify any possible risks, what problems they could cause and finally what steps could be taken to prevent them maturing.

Risk management

Risk management consists of three main activities: identifying potential risks, analysing those risks for probability and impact, and actions taken or planned to mitigate risks should they occur. This needs to be an ongoing activity as new risks may be discovered as the project progresses. The suggestion is to have a project risk register rather than a specific testing one. That way all known project risks (including the testing ones) are visible to everyone. It is a good idea also to split risks out in the register to those which occur in every project and everyone should be aware of. Then also identify any project specific risks that need attention. That way the project specific risks will not get lost in the "regular" risks and run the chance of being missed or their importance understood.

There are also different ways to prioritise risks. The main thing though is to involve others (e.g. the business) as testers will tend to have their own views and these may not match with what the business expect or want.

Risk based testing

A risk-based approach to testing provides proactive opportunities to reduce the levels of product risk, starting in the initial stages of a project. It involves the identification of product risks and their use in guiding test planning and control, specification, preparation and execution of tests. In a risk based approach the risks identified may be used to:

- determine the test techniques to be employed;
- determine the extent of testing to be carried out;
- prioritise testing in an attempt to find the critical defects as early as possible;
- determine whether any non-testing activities could be employed to reduce risk (e.g., providing training to inexperienced designers).

Risk-based testing draws on the collective knowledge and insight of the project stakeholders to determine the risks and the levels of testing required to address those risks.

5.6 Incident Management

Learning Objectives

LO-5.6.1	K1	Recognise the content of an incident report according to the 'Standard for Software Test Documentation' (IEEE Std. 829-1998).
LO-5.6.2	K3	Write an incident report covering the observation of a failure during testing.

Terms

incident logging, incident management, incident report.

What is an incident?

An incident is any event that occurs during testing that requires subsequent investigation or correction. This can equally apply to documentation as well as code. Usually, it signals the mismatch between the actual and expected results of a test (a failure occurs). The cause of this can be one of a number of things:

- a defect in the software;
- a defect in the test (e.g. expected result was wrong);
- the environment was wrong;
- the test was run incorrectly (e.g. entered the wrong input);
- a documentation or specification defect (i.e. what the specification says is wrong).

Purpose of incident reports

These reports have a number of practical uses, in giving feedback about a problem that has been found, to metrics and process improvement. There is a template in IEEE 829-1998 that provides a guide on the contents you could expect to see in a full incident report.

Whilst we can log incidents at any stage throughout the lifecycle, it is advisable to log incidents whenever someone other than the author of the software performs the testing. This is largely because the benefit of a developer logging incidents on his or her code before they hand it to anyone else is vastly outweighed by the cost of doing so. It is much cheaper for a developer to simply fix the problem and retest it than it is for him or her to stop and log the problem before fixing it. There is also a psychological cost to the developer – effectively having to log all the defects in their own code is not a big motivator.

There is a distinct danger that not enough time is spent in logging an incident. It is the tester's responsibility to raise incidents factually and with enough detail for the developer to do their job efficiently. Otherwise we could end up with the situation that the defect cannot be reproduced by the developer so the incident report is returned with a request for more information, or worse still, the incident is ignored. Spending extra time logging sufficient information such that it can be reliably and quickly reproduced will be of great benefit.

Information we should record typically includes:

- test id (the test that failed);
- details of the test environment (e.g. operating system, browsers, etc.);
- id and version of the software under test;
- both actual and expected results (for comparison purposes and to help developers track the defect);
- severity (impact of the failure to the customer/user);
- priority (the urgency of fixing the defect);
- name of the tester or automated test information; and
- any other relevant information needed so that the developer can reproduce and fix the defect.

An incident is basically anything that the developer needs to know or have, in order to reproduce the defect with ease. We should not tell them how to code the changes though!

There might be other information that can be recorded that will help you and your organisation with metrics and monitoring of progress (for example, the effort spent on handling the incident).

Monitoring incidents

Incident reports can be analysed to monitor and improve the test process. For example, if a significant number of incidents reported during system testing turn out to be coding defects that could have been found by component testing then this tells us that the component testing process should be improved.

Reporting incidents

When reporting incidents there are 3 key objectives:

- to give the developer as much information as possible about the incident you have found. There are no secrets in giving information over so the aim should be to give details about everything you did in the first place to find the problem;
- provide the opportunity for the test leaders to track the quality of the application by looking at the numbers and types of incidents, and monitor test progress against plans;
- allow individuals to keep process improvement in their sights and to help them consider how to avoid similar problems in future.

There is a great temptation to report an incident as quickly as possible. The danger though is either not enough or the wrong information could be provided which in turn leads the developer to request more information, put the wrong fix in or not being able to even identify the problem. As testers we need to make sure the developer has all the information relating to an incident. It is said a good tester is one who gets a defect fixed not just finding a defect.

The more defects there are in the software the more defects the testers will have to report. How many can they be reasonably expected to report before this defect reporting time (and future re-testing time) has a significant detrimental impact on the planned testing effort? When estimating test effort it is important to consider how many defects are likely to be found and so how much time will be spent reporting and re-testing them.

Tracking incidents

Incidents should be tracked from inception through the various stages to their final resolution. For any incident logged we must be in a position of knowing its exact status, whether it be waiting for further action, be with a developer for fixing, with the tester for re-testing, or has been re-tested and cleared.

Severity versus priority

It is useful to distinguish severity and priority, because they are different aspects. Severity is related to the impact of a failure caused by a defect. Priority is related to the urgency of fixing a defect. For example, if a defect is holding up a series of automated tests, it could have high priority even if the impact on the user is low.

Incident management tools

Incident management tools store and manage incident reports (bug logs) and help with the management of incidents in numerous ways, such as:

- prioritisation;
- assignment to various people (for fixing, testing or clearing);
- attribution of the status of the incident report (reject, fixed, ready for test, defer to next release).

These tools enable incidents to be monitored and tracked throughout their life. They also often provide data and statistical analysis which helps with process improvement. They are also known as defect tracking tools and can often be found within Test Management Tools.

Session 6

Tool Support for Testing

Contents

- 6.1 Types of Test Tool
- 6.2 Effective Use of Tools: Potential Benefits and Risks
- 6.3 Introducing a Tool into an Organisation

2

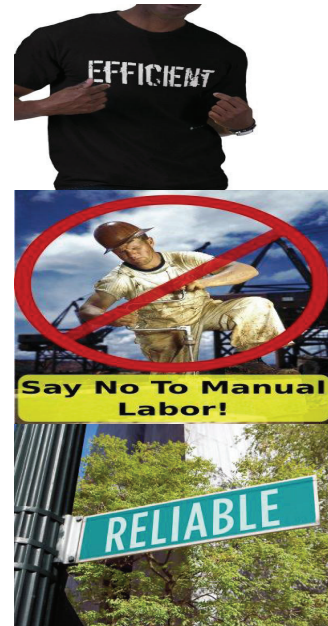
The Use of Tool Support for Testing



3

Purposes of Tool Support for Testing

- improve efficiency
 - automating repetitive tasks
- reduce significant manual testing resource
 - automate static analysis
- test those activities that cannot be achieved manually
 - automate multi-user testing
- increase testing reliability
 - automating large data comparisons or simulating behaviour



4

Testing Tool Classification

Tool information available from:

- Tool lists (e.g. SQE)
- Web sites (e.g. www.stickyminds.com)

management

- test management tools
- requirement management tools
- incident management tools
- configuration management tools

static testing

- review process support tools
- static analysis tools
- modelling tools

test specs

- test design tools
- test data preparation tools

execution

- test execution tools
- test harness/unit test framework tools
- test comparators
- coverage measurement tools
- security tools

performance

- dynamic analysis tools
- performance/load/stress testing tools
- monitoring tools

specific needs

- data quality assessment
- usability...

5

Contents

6.1 Types of Test Tool

6.2 Effective Use of Tools: Potential Benefits and Risks

6.3 Introducing a Tool into an Organisation

6

Potential Benefits and Risks of Using a Tool

BENEFITS

- repetitive work reduced
- consistency
- repeatability
- tests that can't be run manually
- increased coverage
- objective assessment
- ease of access of information about tests or testing

RISKS

- unrealistic expectations
- underestimating time, cost and initial effort
- underestimate in time to achieve significant benefits
- underestimating maintenance effort
- over-reliance on the tool
- vendor problems
- constant re-learning due to intermittent use

7

Contents

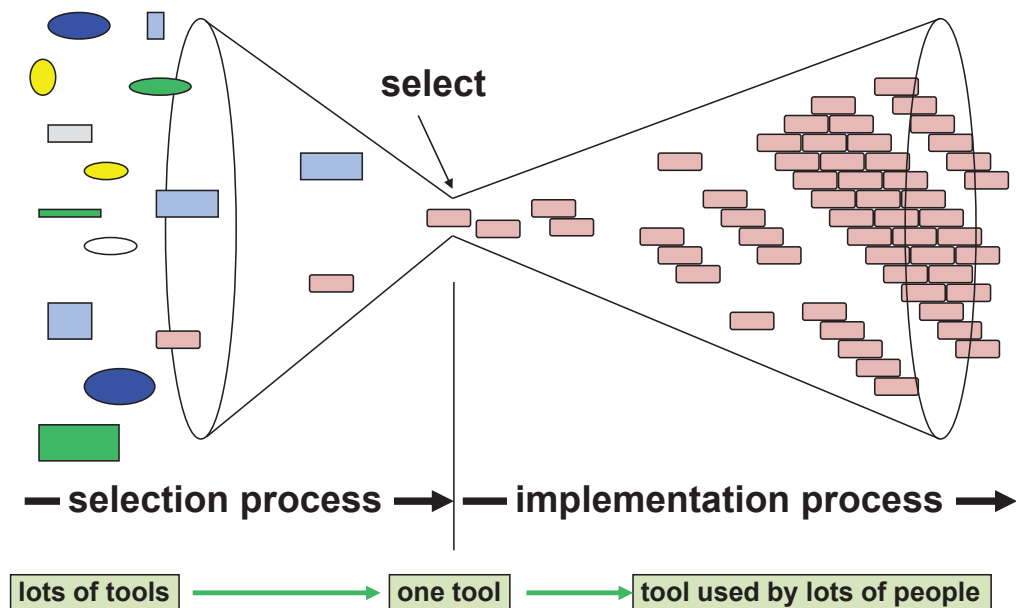
6.1 Types of Test Tool

6.2 Effective Use of Tools: Potential Benefits and Risks

6.3 Introducing a Tool into an Organisation

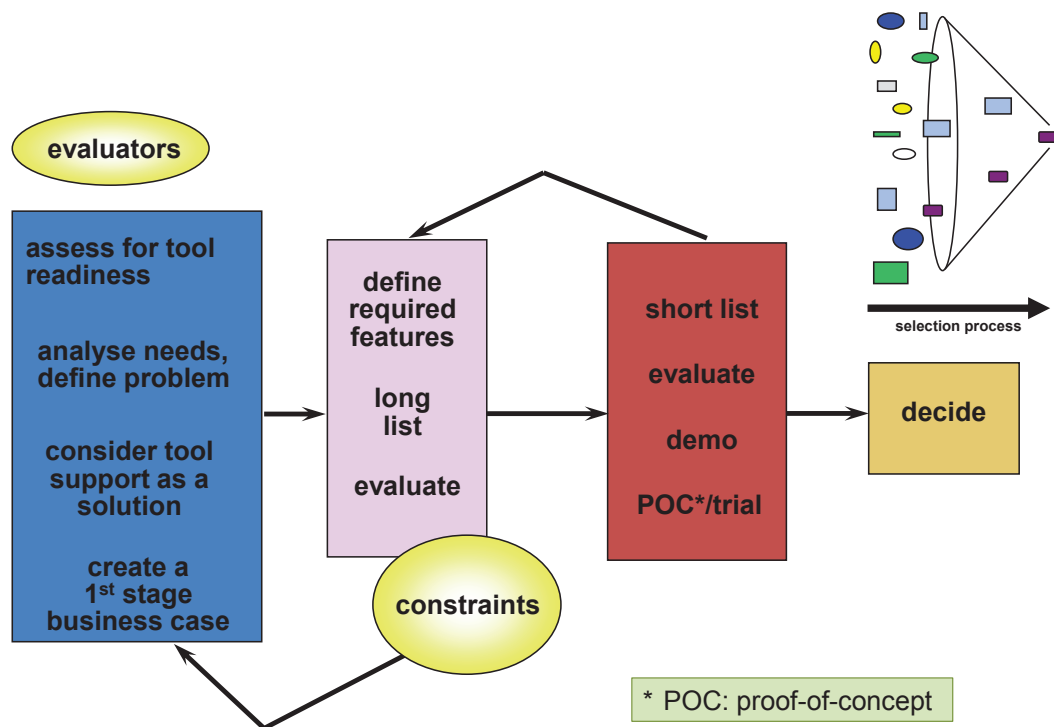
8

Tool Selection and Implementation



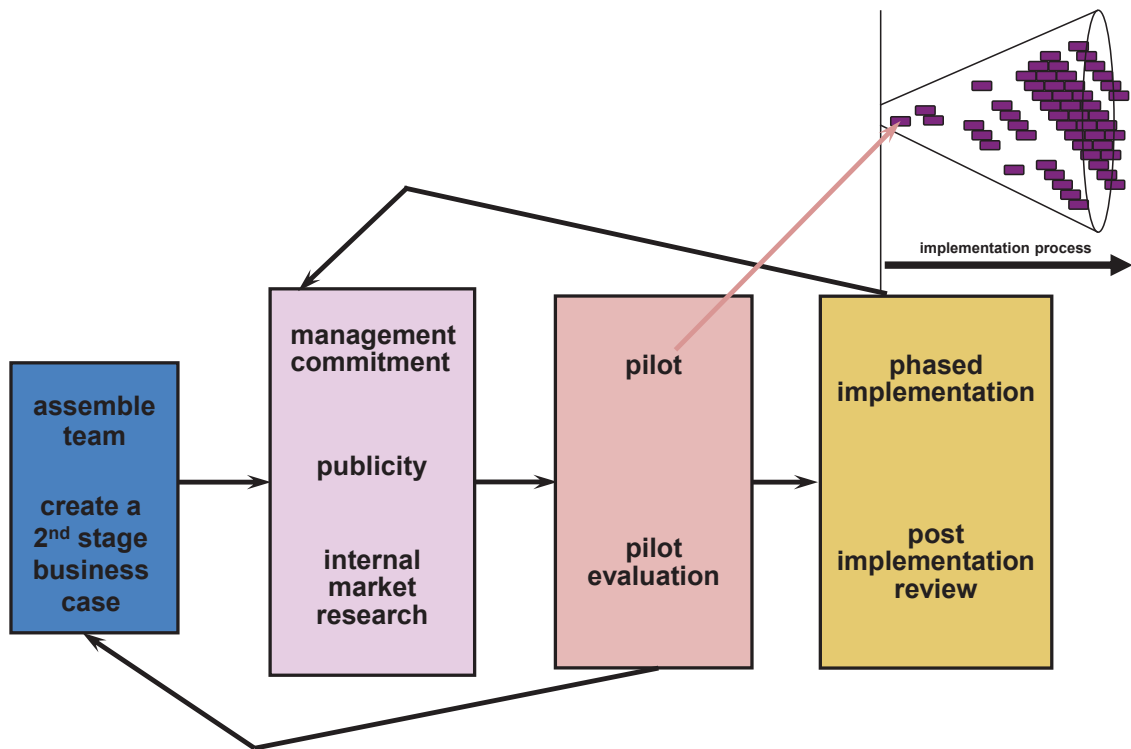
9

The Tool Selection Process



10

The Tool Implementation Process



11

Pilot Project and Implementation

- objectives of the pilot project
 - gain experience in the use of the tool
 - identify changes in test process
 - set internal standards and naming conventions
 - assess costs and achievable benefits
- implementation
 - based on successful pilot
 - needs strong commitment from tool users & managers (overcome resistance, overheads for learning curve)

12

Characteristics of a Pilot Project

Planned	resourced, targets, contingency
Important	full time work, worthwhile tests
Learning	informative, useful, revealing
Objective	quantified, not subjective
Timely	short term, focused

13

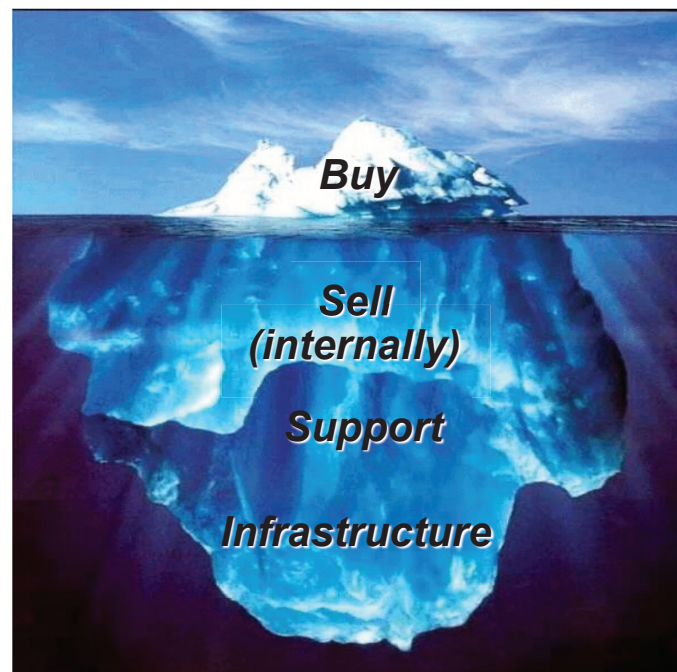
Success Factors for Deployment

Selective	selective, incremental roll-out
Using	adapting/improving processes to use tool
Coaching	coaching & mentoring new users
Criteria	define use criteria/guidelines
Evaluate	evaluate tool use and benefits
Support	providing support for the test team
Share	sharing lessons learnt from all teams

14

Tool Implementation Iceberg

This is an actual photo of an iceberg in Newfoundland. It is estimated to weigh 300,000,000 tons.



15

Summary – Key Points

- what are the 6 categories of tool: **management, static, test specification, execution, performance, misc.**
- what tools are mainly used by developers?
static analysis, modelling, test harness/unit test framework, coverage measurement, dynamic analysis, debugging
- what are the considerations for execution tools?
effort required in scripting for maximum benefit
- what are the key objectives of a pilot project?
gain experience, identify process changes

TOOL SUMMARY

Test Management Tools

- management of testware
 - test plans, specifications, results, incidents
- interface to other tools
 - test execution tools, incident logging and CM
- traceability
 - tests, test results and incidents to source documents
- reports and metrics
 - logging and reporting results,
 - quantitative analysis of tests
 - for process improvement



Special Considerations

this tool needs to interface with other tools or spreadsheets, reports need to be designed well, often required to be centrally located

2

Requirements Management Tools

- automated support for verification and validation of requirements models
 - stores requirement statements
 - ▶ checks for consistency
 - ▶ helps prioritise
 - traceability to design, code, tests
 - ▶ helps manage other documentation
 - ▶ duplicates some test management tool functionality
 - ▶ reports coverage of requirements, functions and features by the tests



3

Incident Management Tools

- also known as defect tracking/bug logging tools
 - commercial or in-house built
- stores, manages & monitors incident reports
 - ability to log information concerning the incident (this is the tester's requirement specification)
 - ▶ priority, status, assignment, description, environment
- provides support for various analysis
 - statistical analysis & data analysis
 - reports often built in and customisable



4

Configuration Management Tools

- provide valuable infra-structure for testing to succeed
 - not strictly testing tools
- what can they do?
 - stores information about versions/builds of software
 - traceability between testware and software
 - assist with change control
 - useful when developing more than one configuration per release



5

Review Tools (Review Process Support Tools)

- mostly “in-house” built
 - spreadsheets, databases and templates
- what can they do?
 - stores information about the review process
 - ▶ comments, defects, effort, changes, metrics
 - provides traceability between
 - ▶ rules and checklists
 - ▶ documents and source code
 - helps with on-line reviews
 - ▶ infrastructure if teams are geographically dispersed



6

Static Analysis Tools (Developer)

- provide information about the quality of software
 - code is examined, not executed
 - helps enforce coding standards and aids understanding of the code
- supports developers, testers and QA teams
 - finds defects before test execution, saves time & money
 - may accelerate and improve process by having less rework
- supports managers
 - objective measures:
e.g. CC* & LOC*
(for risk analysis)



special considerations

may generate lots of messages, warning messages need addressing, filtering of messages is desirable

* CC: cyclomatic complexity
LOC: lines of code

7

Modelling Tools (Developer)

- validate models of the software
 - finding defects in data, state and/or object models
- aids test case generation
 - where test cases are based on the model
- benefits
 - defects are found at the earliest opportunity
 - ▶ saves time, money and accelerates process



8

Test Design Tools

- generate test inputs
 - from a formal specification (test basis)
 - from a CASE repository (design model)
 - from code (e.g. code not covered yet)
- generate “limited” expected outcomes
 - from an Oracle (e.g. existing system)
- benefits
 - save time, increased thoroughness (but can generate too many tests!)



9

Test Data Preparation Tools

- data manipulation
 - selected from existing databases or files
 - created according to some rules
 - edited from other sources
 - safeguards “data protection act” when using live data
- data used during test execution
 - copied and manipulated “live/test” data



10

Test Execution Tools

- interface to the software being tested
 - run tests as though run by a human tester, simulates user interaction, keystrokes, mouse movements
- test scripts in a programmable language
- data, test inputs and expected results held in test repositories
- most often used to automate regression testing
 - can also be useful to record tests during an exploratory test session

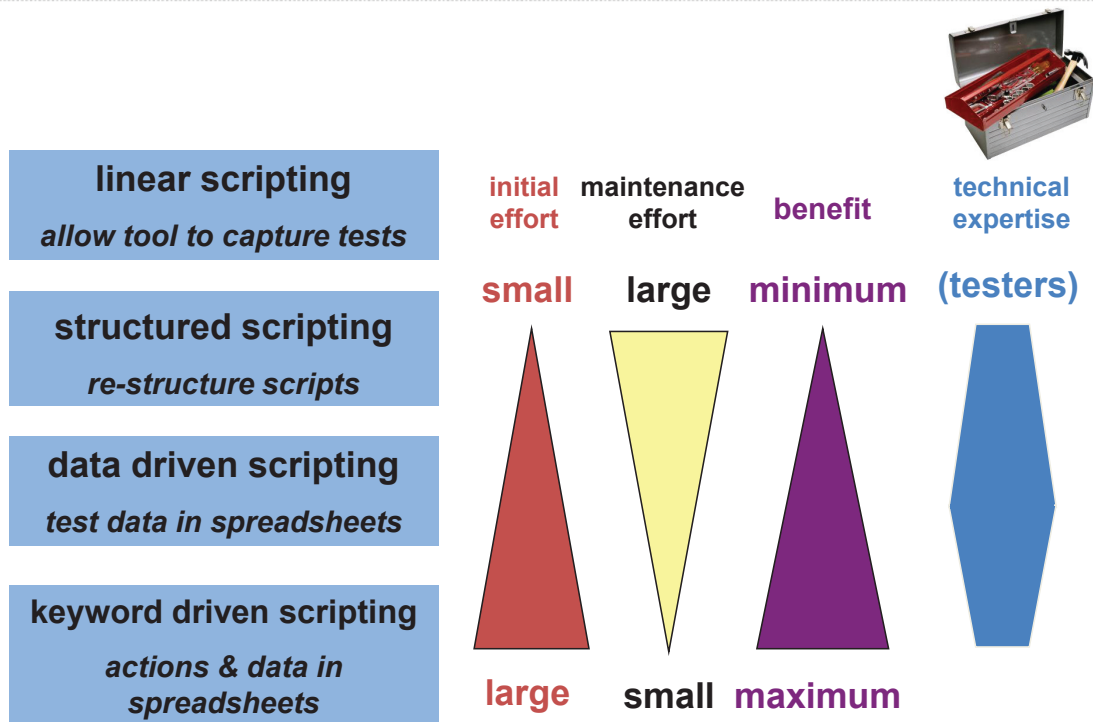


special considerations

significant effort in scripting is required to achieve significant benefits from this type of tool

11

Considerations for Execution Tools



12

Test Comparators

- detect differences between actual test results and expected results
 - screens, characters, bitmaps
 - masking and filtering
- test running tools normally include dynamic comparison capability
- stand-alone comparison tools for files or databases (post execution comparison)
- a test oracle may be used as a test comparison



13

Test Harnesses / Unit Test Framework (Developer)

- used to exercise software that does not have a user interface (yet)
- facilitates testing by simulating an environment in which the test will be run
 - **simulators** (where testing in real environment would be too costly or dangerous)
- aids component testing in parallel with building the code
- may be custom-built
- includes debugging support



14

Coverage Measurement Tools (Developer)

- objective measure of test coverage
 - **tool reports what has and has not been covered by those tests, line by line and summary statistics**
- different types of coverage
 - **statement, decision, branch, condition, LCSAJ, et al**
- intrusive (code is changed)
 - **code is instrumented**
 - **tests are run through the instrumented code**
- non-intrusive (code is not changed)
 - **this is a sampling technique**



15

Security Tools / Security Testing Tools

- security tools
 - e.g. virus checking, firewalls
 - not strictly testing tools but can assist in security testing
- tools that support security testing
 - searching for vulnerabilities
 - ▶ e.g. denial of service attacks, probe for open ports, password files
 - can attack networks, support software, application code, database



16

Dynamic Analysis Tools (Developer)

- finds defects that are only evident when the software is / tests are run
 - provide run-time information on software
 - ▶ allocation, use and de-allocation of resources, e.g. monitoring memory use helps find 'memory leaks'
 - ▶ highlight unassigned pointers or pointer arithmetic defects
- useful when testing middleware
- typically used in component testing and component integration testing



17

Performance Testing Tools

- performance, load and stress tools
 - drive application via user interface or test harness
 - simulates realistic load on the system, application, a database or environment (monitors behaviour)
 - logs number of transactions & response times for selected transactions via user interface
- reports based on logs, graphs of load versus response times



special considerations

expertise is required in the tool and in performance testing techniques to design tests and interpret results

18

Monitoring Tools

- continuously analyse, verify and report...
 - usage of specific system resources
 - warnings of possible service problems
- store information about software
 - helps with traceability
- often used by operations



Note: this is not strictly a testing tool but provides information to assist testing in the identification and analysis of certain types of defects

19

SESSION 6: TOOL SUPPORT FOR TESTING - NOTES

Terms

configuration management tool, coverage tool, data-driven testing, debugging tool, dynamic analysis tool, incident management tool, keyword-driven testing, load testing tool, modelling tool, performance testing tool, probe effect, requirements management tool, review tool, scripting language, security testing tool, security tool, static analysis tool, stress testing tool, test comparator, test data preparation tool, test design tool, test harness, test execution tool, test management tool, unit test framework tool.

From the ISTQB Glossary

configuration management tool: A tool that provides support for the identification and control of configuration items, their status over changes and versions, and the release of baselines consisting of configuration items.

coverage tool: A tool that provides objective measures of what structural elements, e.g., statements, branches have been exercised by a test suite.

data-driven testing: A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools.

debugging tool: A tool used by programmers to reproduce failures, investigate the state of programs and find the corresponding defect. Debuggers enable programmers to execute programs step by step, to halt a program at any program statement and to set and examine program variables.

dynamic analysis tool: A tool that provides run-time information on the state of the software code. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic and to monitor the allocation, use and deallocation of memory and to flag memory leaks.

incident management tool: The process of recognizing, investigating, taking action and disposing of incidents. It involves logging incidents, classifying them and identifying the impact.

keyword-driven testing: A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test.

load testing tool: A tool to support load testing whereby it can simulate increasing load, e.g., numbers of concurrent users and/or transactions within a specified time-period.

modelling tool: A tool that supports the creation, amendment and verification of models of the software or system.

performance testing tool: A tool to support performance testing that usually has two main facilities: load generation and test transaction measurement. Load generation can simulate either multiple users or high volumes of input data. During execution, response time measurements are taken from selected transactions and these are logged. Performance testing tools normally provide reports based on test logs and graphs of load against response times.

probe effect: The effect on the component or system by the measurement instrument when the component or system is being measured, e.g., by a performance testing tool or monitor. For example performance may be slightly worse when performance testing tools are being used.

requirements management tool: A tool that supports the recording of requirements, requirements attributes (e.g., priority, knowledge responsible) and annotation, and facilitates traceability through layers of requirements and requirements change management. Some requirements management tools also provide facilities for static analysis, such as consistency checking and violations to pre-defined requirements rules.

review tool: A tool that provides support to the review process. Typical features include review planning and tracking support, communication support, collaborative reviews and a repository for collecting and reporting of metrics.

scripting language: A programming language in which executable test scripts are written, used by a test execution tool (e.g., a capture/playback tool).

security testing tool: A tool that provides support for testing security characteristics and vulnerabilities.

security tool: A tool that supports operational security.

static analysis tool: A tool that carries out static analysis.

stress testing tool: A tool that supports stress testing.

test comparator: A test tool to perform automated test comparison of actual results with expected results.

test data preparation tool: A type of test tool that enables data to be selected from existing databases or created, generated, manipulated and edited for use in testing.

test design tool: A tool that supports the test design activity by generating test inputs from a specification that may be held in a CASE tool repository, e.g., requirements management tool, from specified test conditions held in the tool itself, or from code.

test execution tool: A type of test tool that is able to execute other software using an automated test script, e.g., capture/playback.

test harness: A test environment comprised of stubs and drivers needed to execute a test.

test management tool: A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, the logging of results, progress tracking, incident management and test reporting.

unit test framework tool: A tool that provides an environment for unit or component testing in which a component can be tested in isolation or with suitable stubs and drivers. It also provides other support for the developer, such as debugging capabilities.

6.1 Types of Test Tool

Learning Objectives

- | | | |
|----------|----|------------------------------------------------------------------------------------------------------------------------------------------------------|
| LO-6.1.1 | K2 | Classify different types of test tools according to their purpose and to the activities of the fundamental test process and the software life cycle. |
| LO-6.1.2 | K2 | Explain the term test tool and the purpose of tool support for testing. |
-

Terms

configuration management tool, coverage tool, debugging tool, dynamic analysis tool, incident management tool, load testing tool, modelling tool, performance testing tool, probe effect, requirements management tool, review tool, security testing tool, security tool, static analysis tool, stress testing tool, test comparator, test data preparation tool, test design tool, test harness, test execution tool, test management tool, unit test framework tool.

6.1.1 Tool Support for Testing

Most testing activities can benefit from tool support though this does not mean that any of the testing activities can be undertaken entirely by a tool. Generally tools help out by making testing activities less error prone, faster, more accurate, or simply easier. The following summarises areas where tool support for testing is typical.

1. Tools that are directly used in testing,
e.g. test execution tools, test data generation tools and result comparison tools.
2. Tools that help in managing the testing process,
e.g. tools used to manage tests, test results, data, requirements, incidents, defects, etc., and for reporting and monitoring test execution.
3. Tools that are used in reconnaissance, or, in simple terms: exploration,
e.g. tools that monitor read and write operations to a disc or a network.
4. General tools that can aid testing,
e.g. a spreadsheet application or a database querying utility.

The purposes of using tools to support testing activities are many and varied, depending on the context. However, they are typically one or more of the following:

- improve the efficiency of test activities by automating repetitive tasks or supporting manual test activities like test planning, test design, test reporting and monitoring;
- automate activities that require significant resources when done manually (e.g., static testing);
- automate activities that cannot be executed manually (e.g., large scale performance testing of client-server applications);
- increase reliability of testing (e.g., by automating large data comparisons or simulating behaviour).
- The term “test frameworks” is also frequently used in the industry, in at least three meanings:
- reusable and extensible testing libraries that can be used to build testing tools (also called a test harnesses);
- a type of design of test automation also called a scripting technique (e.g., data-driven, keyword-driven);
- the overall process of test execution.

For the purpose of this syllabus, the term “test frameworks” is used in its first two meanings as described in section 6.2.2.1.

6.1.2 Test Tool Classification

There are various testing tools that assist and support different aspects of testing. In the syllabus tools are classified by the testing activities that they support. It is important to realise that testing tools can support more than one activity but for the purpose of this course they are classified under the activity that they are most closely associated with.

Some tools are supplied by commercial vendors whereas other tools are created “in-house” or are available as open-source or shareware from the internet.

Some types of test tool can be intrusive, which means that they can affect the actual outcome of the test. For example, the actual timing of an executed test may be different due to the load imposed on the system by the test tool itself. Many coverage measurement tools edit the source code (to insert additional instructions that record the code sections that are executed). The consequence of intrusive tools is called the probe effect.

Tools that are more appropriate for developers (such as those used during coding and component testing) are marked with a “(developer)” in the classifications below.

6.1.3 Tool Support for Management of Testing and Tests

Test management tools

Test management tools help throughout the software development lifecycle. This category covers tool support for test planning and monitoring, but also incident management (defect management) tools. For example, some test management tools help with decomposition of the system functionality into test cases and are able to track and cross-reference from requirements through to test cases and back again. In this way, if a requirement changes it is possible for the test management tool to highlight those test cases that will need to be updated and rerun. Similarly if a test case fails, the tools will be able to highlight the requirement(s) affected. (Note that this traceability capability is duplicated in the more sophisticated Requirement Testing tools.)

Many test management tools are integrated with (or provide an interface to) other testing tools, particularly test running tools. This can be exceedingly helpful since it becomes possible to cause automated tests to be executed from within the test management tool, where you have all the information on the success or failure of the previous tests.

Requirements management tools

Requirements management tools (sometimes known as requirements testing tools) help to keep track, store and analyse requirements. These tools can work on requirement specifications written in a formal structured language or just plain English. Although they cannot help validate requirements (i.e. tell you if the requirements are what the end user actually wants) they can help with verifying the requirements (i.e. checking conformance to standards for requirements specifications).

A word processor can be seen as a very basic requirement-testing tool since one of the functions of these tools is to check grammar, which could lead to ambiguity – a serious source of requirement problems.

Requirements tools can check for consistent use of key terms throughout a specification. They may enable “what-if” scenarios (also known as “animation”). They also allow people to collaborate and communicate with each other concerning requirement statements, so enabling better review of them. In short, the requirements management tools can be very useful in testing, as they provide good support for verification of requirements. The tools can also serve as a useful starting point to derive test conditions.

The more sophisticated requirements tools also provide the ability to manage related documents such as Software Design Specifications and Test Specifications. In doing this they can support traceability of requirements, that is, identify which code modules and which tests

support specific requirements. Note that this capability is duplicated by Test Management tools.

One of the possible pitfalls of type of tool is false confidence. The fact that the tool does not find anything wrong with a requirement specification does not imply it is perfect but someone is likely to see it that way! They do require manual intervention, they are not automatic and they certainly cannot correct all the (potential) defects that they find.

Perhaps the most obvious pitfall is that the requirements have to be written down. Many organisations fail to produce a complete requirement specification and for them this type of tool will have limited value (unless it proves to be the catalyst for more complete requirements specifications).

Incident management tools

Incident management tools store and manage incident reports (bug logs) and help with the management of incidents in numerous ways, such as:

- prioritisation;
- assignment to various people (for fixing, testing or clearing);
- attribution of the status of the incident report (reject, fixed, ready for test, defer to next release).

These tools enable incidents to be monitored and tracked throughout their life. They also often provide data and statistical analysis which helps with process improvement. They are also known as defect tracking tools and can often be found within Test Management Tools.

Configuration management tools

Configuration Management (CM) tools are not really testing tools – however a good CM process within your organisation is essential for the success of testing and development. CM tools are required to keep track of different versions and builds of the software and also the tests.

CM tools are often associated with version control and whilst this is true it is only a small part of what these tools can do.

Configuration Management tools:

- store information about versions of the software, project testware and tool support;
- enable traceability between versions of testware and software;
- understand what has changed from one version to another;
- keep track of developing and implementing different configurations of the system.

Configuration Management tools are much more than “source control” and it is important that all software, hardware and testware come under configuration management and change control.

6.1.4 Tool Support for Static Testing

Review tools (review process support tools)

These tools are mainly in-house built (such as small databases and spreadsheets). The review process support tool stores various information about the review process:

- review comments;
- problems and issues encountered;
- metrics from the review/Inspection process;
- manage references to rules and checklists.

These tools might also help with on-line reviews which is essential for companies who are geographically separated.

Static analysis tools (developer)

Static analysis tools analyse source code without executing it. They are a type of super compiler that will highlight a much wider range of real or potential problems than compilers do. Static analysis tools detect all of certain types of fault much more effectively and cheaply than can be achieved by any other means. For example, they can highlight unreachable code, some infinite loops, use of a variable prior to its definition, and redefinition of a variable without an intervening use. These and many more potential faults can be difficult to see when reading source code but can be picked up within seconds by a static analysis tool.

Such tools also calculate various metrics for the code such as McCabe's cyclomatic complexity, Halstead metrics and many more. These can be used, for example, to direct testing effort to where it is most needed.

Although an extremely valuable type of testing tool, it is one that is not used by many organisations. The pitfalls are more psychological than real, for example, a static analysis tool may highlight something that is not going to cause a failure of the software. This is because it is a static view of the software.

Modelling tools (developer)

Modelling tools are able to validate models of the software. Different models might be validated – for instance database models, state models or object models. Defects and inconsistencies can be found with the models and the tools can help in generating test cases which are based on the model.

As with Static Analysis tools, the major benefit of the modelling tools is that defects can be found early in the lifecycle, this saves time, cost and effort in the development process.

6.1.5 Tool Support for Test Specification

Test design tools

Test design tools help to derive test inputs. They are sometimes referred to as test case generators though this is a rather exaggerated claim. A proper test case includes the expected outcome (i.e. what the result of running the test case should be). No tool will ever be able to generate the expected outcome (other than for the most simple and possibly least needed test cases). Thus we prefer to call them partial test case generators.

Test design tools usually work from a formal specification, an actual user interface or from source code. In the first case the specification has to be in a formal language that the test design tool understands or for some tools a CASE (Computer Aided Software Engineering) tool can hold it. A CASE tool captures much of the information required by the test design tool as the system is being designed and therefore saves the need to re-specify the design information in a different format just for the test tool. Where a test design tool uses the user interface of an application the user interface has to be implemented before the test design tools can be used. It is also a fairly restricted set of test inputs that it can generate since they concentrate on testing the user interface rather than the underlying functionality of the software. This is still useful though. When it is the source code that is used to generate test inputs, it is usually in conjunction with a coverage tool, to identify the branch conditions needed to send a test down a particular path.

Test Data preparation tools

Data preparation tools manipulate existing data or generate new test data. Where new data is generated the tool uses a set of instructions or rules supplied by you that describe the format and content of the data to be generated. For example, if you require a lot of names and addresses to populate a database you would specify the valid set of characters and the maximum and minimum lengths of each field and let the tool generate as many records as you require. The names and addresses it generates will not be sensible English names but they will conform to the rules you laid down and so will be valid for the purposes of testing.

Starting with actual data and manipulating it to ensure data privacy and/or reduce its size can generate more realistic test data.

This type of tool makes it possible to generate large volumes of data (as required for volume, performance and stress testing for example) when it is needed. This makes it more manageable since the large volumes do not necessarily have to be kept since they can be regenerated whenever required. On the downside, the technical set up for complex test data may be rather difficult or at least very tedious.

6.1.6 Tool Support for Test Execution and Logging

Test execution tools

Test execution tools (sometimes known as test running tools) enable tests to be executed automatically and in some cases enable the test outputs to be compared to the expected outputs. They are most often used to automate regression testing and usually offer some form of capture/replay facility to record a test being performed manually so it can then replay the same key strokes. The recording is captured in a test script that can be edited (or written from scratch) and is used by the tool to re-perform the test.

These tools are applicable to test execution at any level: unit, integration, system or acceptance testing. The benefits include faster test execution and unattended test execution reducing manual effort and permitting more tests to be run in less time.

The pitfalls are enormous and have caused as many as half of all test automation projects to fail in the long term. The cost of automating a test case is usually far more (between 2 and 10 times) than the cost of running the same test case manually. The cost of maintaining the automated test cases (updating them to work on new versions of the software) can also become larger than the manual execution cost. It is possible to avoid these pitfalls but it is not necessarily easy to do so. For useful advice with this type of tool, read the book "Software Test Automation: effective use of test execution tools" by Mark Fewster and Dorothy Graham, Addison Wesley, 1999.

Test harness/unit test framework tools (developer)

Not all software can be turned into an executable program. For example, a library function that a programmer may use as a building block within his or her program should be tested separately first. This requires a harness or driver, a separate piece of source code that is used to pass test data into the function and receive the output from it.

At component testing and the early stages of integration testing these are usually custom-built though there are a few commercial tools that can provide some support (though they are likely to be language specific).

At later stages of testing such as system and acceptance testing, harnesses (also called simulators) may be required to simulate hardware systems that are not available or cannot be used until the software is demonstrably shown to be reliable. For example, software that controls some aspect of an aircraft needs to be known to work before it is installed in a real aircraft!

Test comparators

Test execution tools usually offer some form of dynamic comparison facilities that enable the output to the screen during the execution of a test case to be compared with the expected output. However, they are not as good at comparing other types of test outcome such as changes to a database and generated report files. For this a stand-alone comparison tool can be used.

These tools offer vastly improved speed and accuracy over manual methods. They will highlight all differences they find, even the ones you are not interested in unless you can specify some form of mask or filter to hide those expected differences such as dates and times. Specifying masks may not be an easy task.

Coverage measurement tools (developer)

Coverage tools assess how much of the software under test has been exercised by a set of tests. They can do this by a number of methods but the most common is for the tool to instrument the source code in a "pre-compiler pass". This involves the tool inserting new instructions within the original code such that when it is executed the new code writes to a data file recording the fact that it has been executed. After a set of test cases have been executed the tool then examines this data file to determine which parts of the original source code have been executed and (more importantly) which parts have not been executed.

Coverage tools are most commonly used at component test level. For example, statement and/or decision coverage is often a requirement for testing safety-critical or safety-related systems. However, some coverage tools can also measure the coverage of design level constructs such as call trees.

It is important not to set some arbitrary coverage measure as a target without a good understanding of the consequences of doing so. Achieving 100% decision coverage may seem like a good idea but it can be a very expensive goal, and may not be the best testing that could be done in the circumstances. Coverage has to be justified against other means of achieving the desired level of quality of software.

Security testing tools

There are a number of tools that protect systems from external attack, for example firewalls, which are important for any system. These are security tools (rather than security testing tools).

Security testing tools can be used to test security by trying to break into a system, whether or not it is protected by a security tool. The attacks may focus on the network, the support software, the application code or the underlying database.

Features or characteristics of security testing tools include support for:

- identifying viruses;
- detecting intrusions such as denial of service attacks;
- simulating various types of external attack;
- probing for open ports or other externally visible possible points of attack;
- identifying weaknesses in password files and passwords;
- security checks by IT security operational - e.g. for checking integrity of files and intrusion detection can support security testing such as checking results of test attacks.

6.1.7 Tool Support for Performance and Monitoring

Dynamic analysis tools (developer)

Dynamic analysis tools assess the system while the software is running. They monitor the software's use of the computer system's resources such as the allocation, use and de-allocation of memory, read and write operations to a disc and CPU use. Monitoring memory in this way helps detect memory leaks and faulty references to memory such as unassigned pointers and incorrect pointer arithmetic.

A memory leak occurs if a program does not release blocks of memory when it should, so the block has leaked out of the pool of memory blocks available to all programs. Eventually the faulty program will end up owning all of memory; nothing can run, the system hangs up and must be re-booted.

Pointers are references to specific memory locations where data can be stored. Consider the analogy of house addresses in a street: "15 High Street" is a specific location. If the pointer does not contain a valid memory location when it is used this is a defect and is likely to cause a failure eventually that can be difficult to trace back to the cause without the use of a

dynamic analysis tool. Considering the analogy of house addresses: “42 High Street” is probably invalid if there are only 30 houses on High Street!

Performance testing/load testing/stress testing tools

If performance measurement is something you need to do, then a performance testing tool is a must. Such tools are able to provide a lot of very accurate measures of response times, service times and the like. Other tools in this category are able to simulate loads including multiple users, heavy network traffic and database accesses. Although they are not always easy to set up, simulating particular loads is usually much more cost effective than using a lot of people and/or hardware, or “testing” your new web site by public failure to cope!

Monitoring tools

Network monitoring tools may also be considered a type of dynamic analysis tool. These tools monitor computer networks typically while they are in normal use. They will alert the network manager when problems occur such as network failures or performance degradation. These capabilities make them good tools to use during testing of systems that operate on networked systems.

These tools are often used by the operations or support department so they can continuously analyse, verify and report on the usage of specific system resources and as a result they are able to give warnings of potential service problems.

6.1.8 Tool Support for Specific Testing Needs

Data quality assessment

Data is at the centre of some projects such as data conversion/migration projects and applications like data warehouses. Data attributes can vary in terms of criticality and volume. In such contexts, tools need to be employed for data quality assessment to review and verify the data conversion and migration rules to ensure that the processed data is correct, complete and complies to a pre-defined context-specific standard.

Others

It is important to realise that even though the tools have been classified in the categories listed above, certain tools can be specialised to work in certain environments, for example there are specific performance and monitoring tools for web applications and there are specific dynamic analysis tools to test specific security aspects. There are also tools that are used to support usability testing (such as those used to detect accessibility issues, e.g. checking colour combinations used in graphical user interfaces do cause difficulties for people who are colour blind).

Some tool providers specialise in specific application areas, for example, embedded systems and telecommunications.

Many developers and testers write small utilities and macros to help and assist them in the testing that they are doing and these may not fall into any of the categories mentioned. SQL is a common utility for selecting and manipulating data, which the developers might use. Debugging tools used by developers help them locate and properly fix defects that have been found.

6.2 Effective Use of Tools: Potential Benefits and Risks

Learning Objectives:

LO-6.2.1 K2 Summarise the potential benefits and risks of test automation and tool support for testing.

Terms

data-driven testing, keyword-driven testing, scripting language.

6.2.1 Potential Benefits and Risks of Tool Support for Testing (for all tools)

Buying the tool is just the “tip of the iceberg”. In order to achieve significant long term benefits additional effort is often required. Success is not guaranteed and there are often risks associated with purchasing and implementing a tool of any type.

In order to achieve the benefits that are often sought – planning needs to be done as well as setting up a good infrastructure and regime for test automation.

Potential benefits of using tools

There are many and varied potential benefits of using tools to support testing depending on the context and the type of tool. Typical benefits are:

- repetitive manual work is reduced (e.g. entering the same data over and over when doing repeated regression tests, or checking code against coding standards);
- consistency and repeatability (e.g. test data extracted from a database, test execution and comparison);
- objective assessment (e.g. static analysis, coverage and system behaviour);
- ease of access to information about tests, faults and testing that has been performed (e.g. performance and management);
- faster and continuous testing (e.g. unmanned regression tests, dynamic analysis);
- tests that could not be run manually (e.g. significant load testing).

Risks of using tools

Unfortunately using tools also comes with risks; success with tools cannot be guaranteed. We could introduce a tool and find that its impact is the opposite to that which we were expecting. For example, the introduction of a test execution tool may have the expected benefit of reducing the elapsed test execution time but if this is not done well it could result in the elapsed test execution time being increased. Some of the typically risks are listed below.

- Unrealistic expectations for the tool. This can be at any level, but often people are oversold on the tool and expect far too much too soon.
- Underestimating the amount of time it takes to introduce a tool into an organisation. Setup, building an infrastructure and laying a foundation for the tool use takes effort and in some instances a significant amount of time. This is obviously dependent on the size of the system and size of organisation.
- Underestimating the effort required to achieve significant benefits from the tool. One problem that could arise is that the benefits are unlikely to be seen on the project that the tool is first used. If the project is “cost-coded” then often project managers will not pay for tool implementation on their project. . (Cost-coding is a method of grouping individual costs based on their nature or function. Project managers less convinced of the benefits of tool support for testing may not wish to be seen spending in that way.)
- Underestimating the effort required in maintenance of the test assets generated by the tool. Also included here is the effort required to maintain the tool.
- Over-reliance of the tool. People (in general) are the best at finding faults – not tools. It is the test that finds the fault. If people rely solely on the tool then the tester’s creativity is lost.
- Neglecting version control of test assets within the tool.
- Neglecting relationships and interoperability issues between critical tools, such as requirements management tools, version control tools, incident management tools, defect tracking tools and tools from multiple vendors.

- Risk of tool vendor going out of business, retiring the tool, or selling the tool to a different vendor.
- Poor response from vendor for support, upgrades, and defect fixes.
- Risk of suspension of open-source / free tool project.
- Unforeseen problems, such as the inability of the tool to support a new platform.

6.2.2 Special Considerations for Some Types of Tool

Some tools need special attention, particularly when significant benefits are required and to safeguard the tool from becoming “shelfware”.

Test execution tools

Test execution tools (commonly known as capture/replay or regression testing tools) often require effort in achieving long term benefits. Capturing tests/inputs using the tool is a useful first step for a small number of initial tests to familiarise yourself with the tool. A captured script is a linear representation with specific data and actions as part of each script (also called a linear script). However this approach does not scale to large numbers of automated test scripts as the scripts may become unstable when unexpected events occur and they are vulnerable to even small changes to the software being tested. In these cases alternative scripting techniques should generally be used, otherwise maintenance of the automated tests is likely to become unmanageable.

Two of the most common alternative scripting techniques are outlined below.

- Data-driven

This is where the data (input and output) is held externally to the tool (usually in a spreadsheet) – this requires technical expertise and programming skills. This technique uses a generic script for each test and allows testers unfamiliar with the scripting language to create additional test data.

Further enhancements to the basic data-driven approach improve its ability to handle more complex situations. For example, in addition to the data being specified in a separate file, some input data can be generated using algorithms based on configurable parameters at run time (such as the generation for a user-id).

- Keyword driven

This is an extension of the data-driven approach in which the external data file also contains keywords describing the actions to be taken (keywords are also called action words). Testers can then define tests using the keywords independent of the scripting language. The keywords can be tailored to the application being tested (this is also known as a Domain Specific Test Language). Maintenance is lower in effort, but the initial set-up costs are higher.

Technical expertise in the scripting language of the test tool is required for all approaches by one or more test automation specialists but not necessarily all the testers.

Regardless of the scripting technique used, the expected results for each test need to be stored for later comparison.

Static analysis tools

Static analysis tools will quickly and easily analyse code and produce code metrics and a list of potential anomalies and faults. When applied to existing code a large number of messages may be produced. Error message highlight problems that will prevent the code from being compiled successfully or will likely cause the code to fail in operation. Warning messages highlight issues that are not conclusively potential failures and issues that, if addressed, will make maintenance of the code easier. When first introduced, the potentially large number of warning messages can be more of a hindrance than a help. In such cases it is better to adopt a gradual implementation that initially uses filters to exclude the least valuable warning messages.

Test management tools

Test management tools are often used to provide timely information in order to make decisions. These tools should interface smoothly and efficiently with other tools used throughout the test process such as defect tracking tools, spreadsheets, and test execution tools.

Managers should be given information which is useful and timely. Reports produced should be designed and monitored regularly so that they provide benefit to the recipients.

6.3 Introducing a Tool into an Organisation

Learning Objectives:

LO-6.3.1	K1	State the main principles of introducing a tool into an organisation.
LO-6.3.2	K1	State the goals of a proof-of-concept for tool evaluation and a piloting phase for tool implementation.
LO-6.3.3	K1	Recognise that factors other than simply acquiring a tool are required for good tools support.

Terms

No specific terms.

6.3.1 Tool Selection and Implementation

This section gives a brief overview of the main principles of introducing a tool into an organisation as described in the syllabus. More information on tool selection and implementation can be found in Chapters 10 and 11 of the book “Software Test Automation” by Mark Fewster and Dorothy Graham, published by Addison-Wesley, ISBN 0-201-33140-3.

At the beginning it will be important to gain funding for this work and treat it as a project. The 1st stage business case will take you through to deciding which tool to go for. You can't really go beyond that point as you don't know the implementation issues until the tool has been chosen. This will help get buy in from management early on.

Which test activities to automate?

Given the wide range of different types of testing tool there is a wide range of testing activities that can be tool supported (but not necessarily fully automated). When considering tool support, identify all the activities where tool support could be beneficial and prioritise them to identify the most important. Although test running tools are the most popular, this may not be the best place to introduce tool support first.

Testing tool requirements

Testing tools often provide a rich set of functions and features. However, the most functionally rich tool may not be best for any particular situation. Having a tool that matches the existing test process is usually more important. If a tool does not adequately support your test process it could be more of a hindrance than a help.

Tool readiness

Tool readiness refers to a test process that is ready for tool support, i.e. the ease with which test support tools can be implemented. If you have a chaotic testing process it is unlikely that any testing tool will help. The benefits of tools usually depend on a systematic and disciplined test process. If the test process is poor it may be possible to improve it in parallel with the introduction of a testing tool but care should be taken. In our experience about half of the organisations that have bought a test tool have abandoned it. Introducing a testing tool is not an easy thing to do well and if it is not done well there is a strong chance that testing will cost more or be less effective as a result of the tool.

Environment constraints and tool integration

The environment into which it will go will almost certainly influence the choice of a testing tool. Some tools can only work on certain hardware platforms, operating systems or programming languages, and some require their own hardware.

If one or more testing tools are already in use it may be necessary to consider their integration with any new tool. This issue may also affect other types of tool, for example, configuration management tools and CASE (Computer Aided Software Engineering) tools. Some vendors offer a variety of integrated test tools where compatibility between the tools is assured although this does not necessarily mean that if one of their tools suits your needs.

Main principles of introducing a tool into an organisation

After it has been decided which testing activity will offer most benefit from tool support the job of selecting a suitable test tool should be considered. The main considerations are given below.

- Assessment of the organisational maturity, strengths and weaknesses and identification of opportunities for an improved test process supported by tools.
- Evaluation against clear requirements and objective criteria.
- A proof-of-concept, by using a test tool during the evaluation phase to establish whether it performs effectively with the software under test and within the current infrastructure or to identify changes needed to that infrastructure to effectively use the tool.
- Evaluation of the vendor (including training, support and commercial aspects) or service support suppliers in case of non-commercial tools.
- Identification of internal requirements for coaching and mentoring in the use of the tool.
- Evaluation of training needs considering the current test team's test automation skills.
- Estimation of a cost-benefit ratio based on a concrete business case.

It may be necessary to produce a business case just for the selection process, since this will involve spending time (and therefore money). This is the 1st stage business case. When a tool has been selected, the cost of the tool and associated other things (hardware, storage, training, etc.) is included in the business case for purchase.

Implementation (pilot project and roll out)

Once the decision to acquire a tool has been made, a 2nd stage business case may have to be made to plan the implementation of the chosen tool. The "tool champion", someone who is working full-time at least initially to help to implement the tool, needs to have his or her time freed from other work. The learning curve will mean that productivity will be affected at first. These things need to be included in the 2nd stage business case.

The task of implementing should start with a pilot project. The aim of this is to ensure that the tool can be used to achieve the planned benefits. Objectives of the pilot project include:

- learn more detail about the tool;
- see how the tool would fit with existing processes and procedures and identify how they might need to change;
- decide on standard ways of using, managing, storing and maintaining the tool and the test assets (e.g. deciding on naming conventions for files and tests, creating libraries and defining the modularity of test suites);
- assess whether the benefits will be achieved at reasonable cost – as identified in the business case. This would be an opportunity to re-set expectations once the tools have been used.

Once the pilot has been assessed, the tool should only be deployed to the rest of the department/organisation if it the pilot has been deemed successful. Success factors for deployment include:

- rolling out the tool to the rest of the organisation incrementally;
- adapting and improving processes to fit with the use of the tool;

- providing training and coaching/mentoring for new users;
- defining use guidelines;
- implementing a way to gather information about the actual use;
- monitoring tool use and benefits;
- providing support for the test team for a given tool;
- gathering lessons learned from all teams.

Exercise

Test Process Game Instructions

Exercise 1 1

Fundamental Test Process Matching Game

This game can be played in teams of 2 to 5 people, but it could also be played individually. Give each team a copy of the game boards (pages 2 and 3 of this document) as shown below.

Test planning and control		
Test planning		Test control
Test analysis and design		

Test implementation and execution		
Evaluating exit criteria and reporting		
Test closure activities		

Also give each team one set of activity cards. These are on page 4 of this document as shown below.

Identifying necessary test data to support test conditions and test cases	Checking which planned deliverables have been delivered	Re-prioritize tests when an identified risk occurs (e.g. software delivered late)
Reviewing the test basis	Closing incident reports or raising change records for any that remain open	Setting an entry criterion requiring fixes to have been retested by a developer
Assigning resources for the different activities defined	Documenting the acceptance of the system	Comparing actual results with expected results
Closing incident reports or raising change records for any that remain open	Setting an entry criterion requiring fixes to have been retested by a developer	Re-prioritize tests when an identified risk occurs (e.g. software delivered late)
Documenting the acceptance of the system	Comparing actual results with expected results	Reviewing the test basis
Reviewing the test basis as a developer	Closing incident reports or raising change records for any that remain open	Setting an entry criterion requiring fixes to have been retested by a developer
Setting an entry retested by a developer	Documenting the acceptance of the system	Comparing actual results with expected results
Identifying necessary test data to support test conditions and test cases	Checking which planned deliverables have been delivered	Reviewing the test basis another

The individual cards are to be cut out - there are 46 cards. It is a good idea to print both the game boards and the activity cards on thick card and laminate them if possible. Pages 5 to 13 of this document provide the reverse side of the activity cards - including both a sequence number and a version string. The sequence number (e.g. 2_15) gives a set number followed by an index number. These can be used to check that each set is complete and keep the sets separate (course participants may hand them back all muddled).

How to play the game / exercise:

Each team puts the activities under the heading they think is correct. After a fixed time or when the first team declares that they has finished, the results are checked.

Each team checks its own results against the Test Process Game Solution handout (the last page of this document - this is included as a page in one of the pdf documents in the Print Folder) give one of these to each student, as these will serve as a revision aid also.

A prize can be given to the winning team based on which team finishes first, and/or the highest number of correct placings. (It is up to the lecturer to define the rules for winning prizes – but tell them before they play!)

Test implementation and execution

Evaluating exit criteria and reporting

--	--	--

Test closure activities

Creating bi-directional traceability between test basis and test cases.	Scheduling test analysis and design activities	Change the test schedule due to availability of a test environment
Designing and prioritising test cases	Scheduling test implementation, execution and evaluation activities	Comparing actual progress against the plan
Designing the test environment set-up, identifying infrastructure and tools	Select metrics to monitor & control preparation & execution, defect resolution & risk issues	Making decisions based on feedback from monitoring and control activities
Evaluating testability of the test basis and test objects	Verifying the mission of testing	Monitoring testing throughout the project
Identifying and prioritising test conditions and required test data	Analysing lessons learned to determine changes needed for future releases and projects	Reporting the status of testing, including deviations from the plan
Identifying necessary test data to support test conditions and test cases	Checking which planned deliverables have been delivered	Re-prioritise tests when an identified risk occurs (e.g. software delivered late)
Reviewing the test basis	Closing incident reports or raising change records for any that remain open	Setting an entry criterion requiring fixes to have been retested by a developer
Assigning resources for the different activities defined	Documenting the acceptance of the system	Comparing actual results with expected results
Defining test levels and entry and exit criteria	Finalising and archiving testware, test environment and test infrastructure for later use	Creating test suites from the test procedures for efficient execution
Defining the level of detail, structure and templates for test documentation	Handover of testware to maintenance	Developing and prioritising test procedures, creating test data
Defining overall testing approach, including definition of test levels, entry & exit criteria	Using the information gathered to improve test maturity	Developing, implementing and prioritising test cases
Determining the scope and risks, and identifying the objectives of testing	Optionally, preparing test harnesses and writing automated test scripts	Executing test procedures manually or using test execution tools
Integrate testing into lifecycle activities: acquisition, supply, development, operation, maint.	Repeating test activities as a result of action taken for each discrepancy	Logging the outcome of test execution and record identities and versions
Making decisions about what to test and what roles will perform the test activities	Reporting discrepancies as incidents and analysing them	Verifying that the test environment has been set up correctly
Assessing if more tests are needed or if the exit criteria should be changed	Writing a test summary report for stakeholders	Verifying and updating bi-directional traceability between the test basis and test cases
Checking test logs against the exit criteria		

STF151101	1_1	STF151101	1_2	STF151101	1_3
STF151101	1_4	STF151101	1_5	STF151101	1_6
STF151101	1_7	STF151101	1_8	STF151101	1_9
STF151101	1_10	STF151101	1_11	STF151101	1_12
STF151101	1_13	STF151101	1_14	STF151101	1_15
STF151101	1_16	STF151101	1_17	STF151101	1_18
STF151101	1_19	STF151101	1_20	STF151101	1_21
STF151101	1_22	STF151101	1_23	STF151101	1_24
STF151101	1_25	STF151101	1_26	STF151101	1_27
STF151101	1_28	STF151101	1_29	STF151101	1_30
STF151101	1_31	STF151101	1_32	STF151101	1_33
STF151101	1_34	STF151101	1_35	STF151101	1_36
STF151101	1_37	STF151101	1_38	STF151101	1_39
STF151101	1_40	STF151101	1_41	STF151101	1_42
STF151101	1_43	STF151101	1_44	STF151101	1_45
				STF151101	1_46

STF151101	2_1	STF151101	2_2	STF151101	2_3
STF151101	2_4	STF151101	2_5	STF151101	2_6
STF151101	2_7	STF151101	2_8	STF151101	2_9
STF151101	2_10	STF151101	2_11	STF151101	2_12
STF151101	2_13	STF151101	2_14	STF151101	2_15
STF151101	2_16	STF151101	2_17	STF151101	2_18
STF151101	2_19	STF151101	2_20	STF151101	2_21
STF151101	2_22	STF151101	2_23	STF151101	2_24
STF151101	2_25	STF151101	2_26	STF151101	2_27
STF151101	2_28	STF151101	2_29	STF151101	2_30
STF151101	2_31	STF151101	2_32	STF151101	2_33
STF151101	2_34	STF151101	2_35	STF151101	2_36
STF151101	2_37	STF151101	2_38	STF151101	2_39
STF151101	2_40	STF151101	2_41	STF151101	2_42
STF151101	2_43	STF151101	2_44	STF151101	2_45
				STF151101	2_46

STF151101	3_1	STF151101	3_2	STF151101	3_3
STF151101	3_4	STF151101	3_5	STF151101	3_6
STF151101	3_7	STF151101	3_8	STF151101	3_9
STF151101	3_10	STF151101	3_11	STF151101	3_12
STF151101	3_13	STF151101	3_14	STF151101	3_15
STF151101	3_16	STF151101	3_17	STF151101	3_18
STF151101	3_19	STF151101	3_20	STF151101	3_21
STF151101	3_22	STF151101	3_23	STF151101	3_24
STF151101	3_25	STF151101	3_26	STF151101	3_27
STF151101	3_28	STF151101	3_29	STF151101	3_30
STF151101	3_31	STF151101	3_32	STF151101	3_33
STF151101	3_34	STF151101	3_35	STF151101	3_36
STF151101	3_37	STF151101	3_38	STF151101	3_39
STF151101	3_40	STF151101	3_41	STF151101	3_42
STF151101	3_43	STF151101	3_44	STF151101	3_45
				STF151101	3_46

STF151101	4_1	STF151101	4_2	STF151101	4_3
STF151101	4_4	STF151101	4_5	STF151101	4_6
STF151101	4_7	STF151101	4_8	STF151101	4_9
STF151101	4_10	STF151101	4_11	STF151101	4_12
STF151101	4_13	STF151101	4_14	STF151101	4_15
STF151101	4_16	STF151101	4_17	STF151101	4_18
STF151101	4_19	STF151101	4_20	STF151101	4_21
STF151101	4_22	STF151101	4_23	STF151101	4_24
STF151101	4_25	STF151101	4_26	STF151101	4_27
STF151101	4_28	STF151101	4_29	STF151101	4_30
STF151101	4_31	STF151101	4_32	STF151101	4_33
STF151101	4_34	STF151101	4_35	STF151101	4_36
STF151101	4_37	STF151101	4_38	STF151101	4_39
STF151101	4_40	STF151101	4_41	STF151101	4_42
STF151101	4_43	STF151101	4_44	STF151101	4_45
				STF151101	4_46

STF151101	5_1	STF151101	5_2	STF151101	5_3
STF151101	5_4	STF151101	5_5	STF151101	5_6
STF151101	5_7	STF151101	5_8	STF151101	5_9
STF151101	5_10	STF151101	5_11	STF151101	5_12
STF151101	5_13	STF151101	5_14	STF151101	5_15
STF151101	5_16	STF151101	5_17	STF151101	5_18
STF151101	5_19	STF151101	5_20	STF151101	5_21
STF151101	5_22	STF151101	5_23	STF151101	5_24
STF151101	5_25	STF151101	5_26	STF151101	5_27
STF151101	5_28	STF151101	5_29	STF151101	5_30
STF151101	5_31	STF151101	5_32	STF151101	5_33
STF151101	5_34	STF151101	5_35	STF151101	5_36
STF151101	5_37	STF151101	5_38	STF151101	5_39
STF151101	5_40	STF151101	5_41	STF151101	5_42
STF151101	5_43	STF151101	5_44	STF151101	5_45
				STF151101	5_46

STF151101	6_1	STF151101	6_2	STF151101	6_3
STF151101	6_4	STF151101	6_5	STF151101	6_6
STF151101	6_7	STF151101	6_8	STF151101	6_9
STF151101	6_10	STF151101	6_11	STF151101	6_12
STF151101	6_13	STF151101	6_14	STF151101	6_15
STF151101	6_16	STF151101	6_17	STF151101	6_18
STF151101	6_19	STF151101	6_20	STF151101	6_21
STF151101	6_22	STF151101	6_23	STF151101	6_24
STF151101	6_25	STF151101	6_26	STF151101	6_27
STF151101	6_28	STF151101	6_29	STF151101	6_30
STF151101	6_31	STF151101	6_32	STF151101	6_33
STF151101	6_34	STF151101	6_35	STF151101	6_36
STF151101	6_37	STF151101	6_38	STF151101	6_39
STF151101	6_40	STF151101	6_41	STF151101	6_42
STF151101	6_43	STF151101	6_44	STF151101	6_45
				STF151101	6_46

STF151101	7_1	STF151101	7_2	STF151101	7_3
STF151101	7_4	STF151101	7_5	STF151101	7_6
STF151101	7_7	STF151101	7_8	STF151101	7_9
STF151101	7_10	STF151101	7_11	STF151101	7_12
STF151101	7_13	STF151101	7_14	STF151101	7_15
STF151101	7_16	STF151101	7_17	STF151101	7_18
STF151101	7_19	STF151101	7_20	STF151101	7_21
STF151101	7_22	STF151101	7_23	STF151101	7_24
STF151101	7_25	STF151101	7_26	STF151101	7_27
STF151101	7_28	STF151101	7_29	STF151101	7_30
STF151101	7_31	STF151101	7_32	STF151101	7_33
STF151101	7_34	STF151101	7_35	STF151101	7_36
STF151101	7_37	STF151101	7_38	STF151101	7_39
STF151101	7_40	STF151101	7_41	STF151101	7_42
STF151101	7_43	STF151101	7_44	STF151101	7_45
				STF151101	7_46

STF151101	8_1	STF151101	8_2	STF151101	8_3
STF151101	8_4	STF151101	8_5	STF151101	8_6
STF151101	8_7	STF151101	8_8	STF151101	8_9
STF151101	8_10	STF151101	8_11	STF151101	8_12
STF151101	8_13	STF151101	8_14	STF151101	8_15
STF151101	8_16	STF151101	8_17	STF151101	8_18
STF151101	8_19	STF151101	8_20	STF151101	8_21
STF151101	8_22	STF151101	8_23	STF151101	8_24
STF151101	8_25	STF151101	8_26	STF151101	8_27
STF151101	8_28	STF151101	8_29	STF151101	8_30
STF151101	8_31	STF151101	8_32	STF151101	8_33
STF151101	8_34	STF151101	8_35	STF151101	8_36
STF151101	8_37	STF151101	8_38	STF151101	8_39
STF151101	8_40	STF151101	8_41	STF151101	8_42
STF151101	8_43	STF151101	8_44	STF151101	8_45
				STF151101	8_46

STF151101	9_1	STF151101	9_2	STF151101	9_3
STF151101	9_4	STF151101	9_5	STF151101	9_6
STF151101	9_7	STF151101	9_8	STF151101	9_9
STF151101	9_10	STF151101	9_11	STF151101	9_12
STF151101	9_13	STF151101	9_14	STF151101	9_15
STF151101	9_16	STF151101	9_17	STF151101	9_18
STF151101	9_19	STF151101	9_20	STF151101	9_21
STF151101	9_22	STF151101	9_23	STF151101	9_24
STF151101	9_25	STF151101	9_26	STF151101	9_27
STF151101	9_28	STF151101	9_29	STF151101	9_30
STF151101	9_31	STF151101	9_32	STF151101	9_33
STF151101	9_34	STF151101	9_35	STF151101	9_36
STF151101	9_37	STF151101	9_38	STF151101	9_39
STF151101	9_40	STF151101	9_41	STF151101	9_42
STF151101	9_43	STF151101	9_44	STF151101	9_45
				STF151101	9_46

Match statements with test levels

In the table below, enter the numbered statements listed in the table at the bottom of the page. We have done three to get you started.

	Test Levels			
	Component Test	Integration Test	System Test	Acceptance Test
Objective of the test	5) Find defects in logic of the code, detailed boundary checking.			
Who does the testing?		3) Developer or Tester.		
Test Basis			11) Requirements, risks, use cases and functional design specification.	

Place the rest of the following entries into the table above. You can use the words or the numbers (✓ = already placed in the table).

	1) Program specification, detailed design, data model
	2) Developer or Buddy Developer
✓	3) Developer or Tester
	4) Find defects in interactions between components
✓	5) Find defects in logic of the code, detailed boundary checking
	6) Find defects in end-to-end functionality
	7) Give confidence that the system is ready to deploy
	8) Architectural design of how the code modules fit together
	9) Independent test team (usually)
	10) Requirements, risks and acceptance criteria
✓	11) Requirements, risks, use cases and functional design specification
	12) Users, customers, stakeholders

This list is not exhaustive; it gives some examples only.

Test Targets

Exercise 2_2

Match test targets with test levels

In the table below, enter the numbered statements listed in the table at the bottom of the page. We have done two to get you started.

Test Type (Target)	Test Levels			
	Component Test	Integration Test	System Test	Acceptance Test
Functional	2) Testing of field level input for each unit or program.			
Non-Functional (Quality Characteristics)				11) Testing that the system is usable from a business perspective.
Structural			5) Menu option coverage, navigation coverage.	

Place the rest of the following entries into the table above. You can use the words or the numbers (✓ = already placed in the table).

1)	Coverage of business rules
✓ 2)	Testing of field level input for each unit or program
3)	Testing how fast information is transferred between components
4)	End to end testing of the whole system
✓ 5)	Menu option coverage, navigation coverage
6)	Module call coverage, interface coverage
7)	Testing reliability of the new system
8)	Robustness, look for memory leaks, resource issues and maintainability
9)	Statement coverage, decision outcome coverage
10)	Testing to make sure the system does what it says it should do from a business perspective
✓ 11)	Testing that the system is usable from a business perspective
12)	Testing the transfer of information between modules and systems

This list is not exhaustive; it gives some examples only.

Reviews Exercise

Having worked through the characteristics of each type of review (as per the syllabus), work in groups of 2 or 3 and list the advantages of the various types of review:

Advantages of an Informal Review**Advantages of a Walkthrough****Advantages of a Technical Review****Advantages of an Inspection**

Black Box

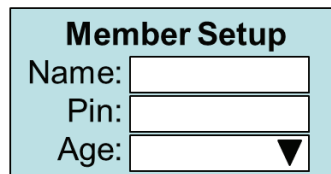
Exercise 4_1

Analyse the test basis

Read through the specification below and determine which parts of the specification need further explanation and which parts are ambiguous or are un-testable.

Specification

A new hard-drive recording system is required for households with up to 6 people. Each new household member will be set up with a unique name (4 – 8 characters), a pin number (4 digits) and an age category is chosen from a list (0-11, 12-17, 18+).



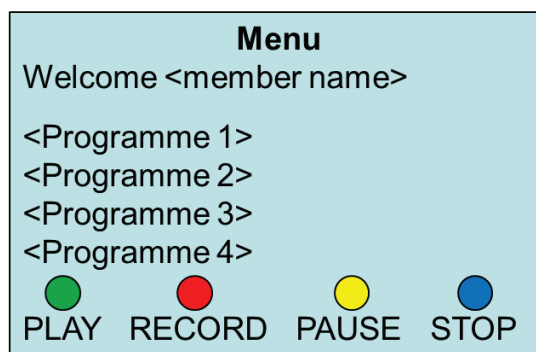
Member Setup

Name:

Pin:

Age:

The age category of the member restricts the viewing for that particular member. 0-11 years can only watch “child” programmes, 12-17 years can watch “child” and “teenager” programmes only and 18+ years can watch all programmes (including “adult”). These restrictions limited to systems being used in Europe. Upon login the member is presented with a menu screen and programmes displayed (that relate to that member's age category) for selection.



Menu

Welcome <member name>

<Programme 1>

<Programme 2>

<Programme 3>

<Programme 4>

PLAY RECORD PAUSE STOP

Once a member is logged onto the system they can watch or record the allowable programme by using the buttons on the remote. When the member selects a programme they can either play or record by pressing the relevant button, a green or red light will display showing the status of the programme. Pressing the “stop” button will return the member to the menu screen. The programme can be paused (when playing or recording) by pressing the “pause” button. The relevant light will flash indicating that the programme has been paused. To resume play or record then the “pause” button is pressed again. When the programme is in a record/pause state it can be stopped by pressing the “stop” button.

Equivalence Partitioning & Boundary Value Analysis

Exercise 4_2

For each of the following examples identify the equivalence partitions and boundary values where possible. (Use the 2-value BVA approach.) Note any questions or assumptions you have made with each of the scenarios. The first one is completed already as an example.

Specification	Equivalence Partitions and Boundary Values
The cardholders name on a credit card processing payment screen can accept up to 32 characters. It cannot be left blank. The field can only accept the following set of characters: A to Z inclusive, a to z inclusive and space.	<p>Cardholder Name: number of characters</p> <p>Invalid Valid Invalid</p> <p>Valid: A-Z a-z space Invalid: any other characters</p>
A financial company decides to offer their customers a credit card based on annual salary. If the customer earns 35,000 euros or more, then they will be accepted for a gold card, if they earn 60,000 euros or more then they will receive platinum, otherwise they will be rejected for the offer.	
In order to pass an exam you need to achieve 60% or more. However, if you achieve 80% or more you will be awarded a distinction.	
A thermostat display shows the temperature to the nearest degree Celsius from -10 to 30. When the temperature is below zero then the message 'Too cold' is displayed. When the temperature exceeds 20 degrees the message 'Too Hot' is displayed.	

Exercise 4_2 Equivalence Partitioning & Boundary Value Analysis

Specification	Equivalence Partitions and Boundary Values
Phone calls are charged according to the time of day the call is made. If a call is made from 08:00 then it will be charged at 1 euro per minute. If it is made from 13:00 calls are charged at 1.50 euro per minute; calls from 17:00 to 22:59 inclusive are charged at 2 euro per minute. All other times will incur no charge.	
An on-line system charges the following shipping costs. For orders of up to 3 books the shipping cost is 5% of the total cost. Orders of 4 to 6 books inclusive are charged a 4% shipping cost and orders of 7 or more books are charged a 3% shipping cost.	
A barman will serve you alcohol if you are at least 18 years of age. Anyone under this age will be offered a soft drink only.	
The types of loyalty card available will be determined by the points accrued on the reward scheme. All customers start on a blue card. After accruing 25,000 points they will receive a red card and after a further 30,000 points they will be eligible for a silver card.	
An application requires a telephone number comprising only digits to be entered. A minimum of 6 digits and a maximum of 18 digits are accepted.	

Decision Table

Exercise 4_3

Scenario

A simple mobile phone will connect calls but only if the number is valid. Some of the calls will be chargeable, whilst others will be free (e.g. emergency number). However if the call is chargeable then a check to see if the account is in credit needs to be done. If the account has credit and the call is chargeable then the account is debited.

Exercise

Step 1: Complete the decision table for the above scenario.

Condition								
Action								
Tags:								

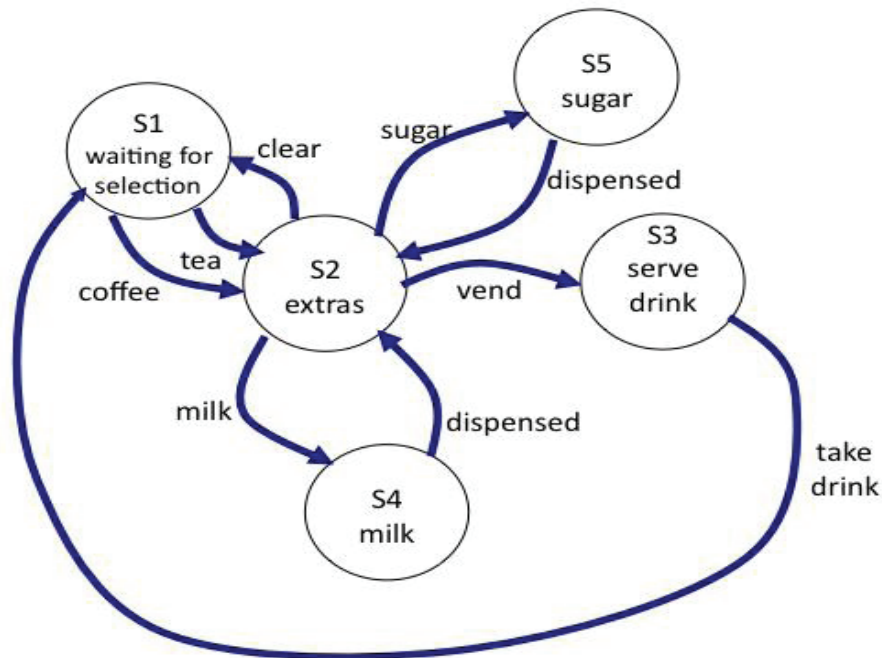
Step 2: Can the decision table be rationalised? Which columns could be removed and why?

Step 3: From the rationalised decision table, choose your top 2 tests, create the test cases and explain your choice.

State Transition

Exercise 4_4

The following simplified State Transition diagram shows the process for ordering a cup of tea or coffee from a vending machine.



Write out test cases using the tags to show the events and the states you go through (e.g. S1 – tea - S2 – sugar – S5 etc.) for the following:

- A test case for a **typical transaction**.
- A test case **that will go through every state**.
- A test case **that will exercise every transition**.
- Complete the following State Table and identify any invalid transitions.

	tea	coffee	clear	sugar	milk	vend	dispensed	take drink
S1 selection								
S2 extras								
S3 serve drink								
S4 milk								
S5 sugar								

1. For the following:

```

IF you buy a cheap-day return ticket THEN
    catch a train after 9.30am
ELSE
    catch any train
ENDIF
Buy a coffee
Read the newspaper
Enjoy the train journey
    
```

a) What is the minimum number of test cases that are required to achieve 100% Statement Coverage?	
b) What is the minimum number of test cases that are required to achieve 100% Decision Coverage?	
c) How much decision coverage have I achieved if I buy a "cheap-day-return" ticket only?	

2. For the following pseudo code:

```

Read A
Read B
IF B = A THEN
    Print "they are the same"
ELSE
    Print "they are different"
ENDIF
Print "End of processing"
    
```

Generate a minimum set of tests (values of A and B) that achieve 100% statement and 100% decision coverage

3. For the following :

```

IF the vending machine is not working THEN
    call repair centre to fix
ELSE
    Insert money
    WHILE there is not enough money
        Display message "insert money"
        Insert more money
    ENDWHILE
    Select a drink
    Wait for drink to be dispensed
    Enjoy the drink!
    Collect any change
ENDIF
    
```

a) Calculate the minimum number of tests required to achieve 100% statement coverage	
b) Calculate the minimum number of tests required to achieve 100% decision coverage	

Exercise 4_5

Structural Techniques

4. Given the following pseudo code:

```

Read P
Read Q
IF P+Q > 100 THEN
    Print "Large"
ELSE
    IF P+Q > 50 THEN
        Print "Medium"
    ENDIF
ENDIF

```

a) What is the minimum number of test cases that are required to achieve 100% Statement Coverage:	
b) What is the minimum number of test cases that are required to achieve 100% Decision Coverage	
c) Provide values for P and Q that will achieve 100% Statement Coverage	
d) Will these same values achieve 100% Decision Coverage	

5. For the following pseudo code:

```

Read (Gross Pay)
Read (Allowances)
Taxable Pay = Gross Pay - Allowances
IF Taxable Pay > 30,000 THEN
    Tax Due = Taxable Pay * 40%
ELSE
    IF Taxable Pay > 15,000 THEN
        Tax Due = Taxable Pay * 23%
    ELSE
        Tax Due = Taxable Pay * 10%
    ENDIF
ENDIF

```

a) What is the minimum number of test cases that are required to achieve 100% Statement Coverage:	
b) What values are required for Gross Pay and Allowances to achieve 100% Decision Coverage?	

Electronic slider puzzle game

This application allows you to play the well known slider puzzle on your computer. After opening the application, click on “New Game” and you will see the game start by the numbers moving around the grid to new positions.

The object of the game is to move the numbers back to their original place in order, starting with number 1 in the top left hand corner and the blank in the bottom right hand corner. You can only move a number next to a blank field. To move it, click on that number and continue until all the numbers are in their correct sequence.

Exercise

Complete the game and identify up to three incidents, using the incident forms provided. Observe what happens when someone else is playing (i.e. testing) the game, and make suggestions for other things to try, even if you haven’t been able to test it directly yourself.

When filling in the form, complete what you can and decide what the severity and priority should be, based on your own judgement.

Hint: we know of one major bug already!

Exercise 5_1

Incident

Incident Report

Incident ID:		Test ID:		Software Version:	
Incident Description. Include actual and expected results:				Severity: 1 = Critical or Testing Stopped 2 = Major or Test Impeded 3 = Minor 4 = Cosmetic / Query / Comment Priority: High Medium Low Attachments:	
Name:		Date:			
Investigation results:				Cause Code:	
Name:				Date:	
Resolution:				Status: Open In development In testing Closed	
Name:				Date:	
Closed By:				Date:	

Incident

Exercise 5_1

Incident Report

Incident ID:	Test ID:	Software Version:
Incident Description. Include actual and expected results:		Severity: 1 = Critical or Testing Stopped 2 = Major or Test Impeded 3 = Minor 4 = Cosmetic / Query / Comment Priority: High Medium Low Attachments:
Name:	Date:	
Investigation results:		Cause Code:
Name:	Date:	
Resolution:		Status: Open In development In testing Closed
Name:	Date:	
Closed By:	Date:	

Exercise 5_1

Incident

Incident Report

Incident ID:		Test ID:	Software Version:
Incident Description. Include actual and expected results:		Severity: 1 = Critical or Testing Stopped 2 = Major or Test Impeded 3 = Minor 4 = Cosmetic / Query / Comment Priority: High Medium Low Attachments:	
Name: _____ Date: _____			
Investigation results:		Cause Code:	
Name: _____ Date: _____			
Resolution:		Status: Open In development In testing Closed	
Name: _____ Date: _____			
Closed By:		Date:	

**Test
Management
Tools**

**Requirements
Management
Tools**

**Incident
Management
Tools**

**Configuration
management
Tools**

**Review
Tools**

**Static Analysis
Tools**

**Modelling
Tools**

**Test Design
Tools**

Exercise 6_1

Tool Pairs

<div>Tool Type</div> <div>STF151030Card 1_1</div>	<div>Tool Type</div> <div>STF151030Card 1_2</div>
<div>Tool Type</div> <div>STF151030Card 1_3</div>	<div>Tool Type</div> <div>STF151030Card 1_4</div>
<div>Tool Type</div> <div>STF151030Card 1_5</div>	<div>Tool Type</div> <div>STF151030Card 1_6</div>
<div>Tool Type</div> <div>STF151030Card 1_7</div>	<div>Tool Type</div> <div>STF151030Card 1_8</div>

**Test Data
Preparation
Tools**

**Test Execution
Tools**

**Test
Comparators**

**Test
Harnesses/Unit
Test Framework**

**Coverage
Measurement
Tools**

**Security
Tools**

**Dynamic
Analysis Tools**

**Performance
Testing Tools**

Exercise 6_1

Tool Pairs

<div>Tool Type</div> <div>STF151030Card 1_9</div>	<div>Tool Type</div> <div>STF151030Card 1_10</div>
<div>Tool Type</div> <div>STF151030Card 1_11</div>	<div>Tool Type</div> <div>STF151030Card 1_12</div>
<div>Tool Type</div> <div>STF151030Card 1_13</div>	<div>Tool Type</div> <div>STF151030Card 1_14</div>
<div>Tool Type</div> <div>STF151030Card 1_15</div>	<div>Tool Type</div> <div>STF151030Card 1_16</div>

Tool Pairs

Exercise 6_1

- **management of testware (test plans, specifications, etc.)**
 - **interfaces to other tools**
 - e.g. test execution, incident logging and CM
 - **traceability of test objects to requirement specifications**
- **also known as defect tracking/bug logging tools**
 - commercial or in-house built
 - **stores, manages & monitors incident reports**
 - ability to log information concerning the incident
 - **provides support for various analysis**
- **mostly “in-house” built**
 - spreadsheets, databases and templates
 - **what can they do?**
 - stores information about the review process
 - comments, defects, effort, changes, metrics
- **validate models of the software**
 - finding defects in data, state and/or object models
 - **aids test case generation**
 - where test cases are based on the model
- **automated support for verification and validation of requirements models**
 - stores requirement statements
 - checks for consistency
 - helps prioritise
 - **traceability to design, code, tests**
- **provides valuable infra-structure for testing to succeed**
 - not strictly testing tools
 - **what can they do?**
 - stores information about versions/builds of software
 - traceability between testware and software
- **provides information about the quality of software**
 - code is examined, not executed
 - **supports developers, testers and QA teams**
 - finds defects before test execution
 - saves time & money
- **generate test inputs**
 - from a formal specification
 - from a CASE repository (design model)
 - from code (e.g. code not covered yet)
 - **generate “limited” expected outcomes**

Exercise 6_1

Tool Pairs

<div>Tool Definition</div> <div>STF151030Card 1_1</div>	<div>Tool Definition</div> <div>STF151030Card 1_2</div>
<div>Tool Definition</div> <div>STF151030Card 1_3</div>	<div>Tool Definition</div> <div>STF151030Card 1_4</div>
<div>Tool Definition</div> <div>STF151030Card 1_5</div>	<div>Tool Definition</div> <div>STF151030Card 1_6</div>
<div>Tool Definition</div> <div>STF151030Card 1_7</div>	<div>Tool Definition</div> <div>STF151030Card 1_8</div>

Tool Pairs

Exercise 6_1

- **data manipulation**
 - selected from existing databases or files
 - created according to some rules
 - edited from other sources
 - safeguards “data protection act” if using live-data
- **detect differences between actual test results and expected results**
 - screens, characters, bitmaps
 - masking and filtering
- **objective measure of test coverage**
 - tool reports what has and has not been covered by those tests, line by line and summary statistics
- **different types of coverage**
 - statement, branch, condition, LCSAJ, etc.
- **helps find defects that are only evident when the software is run**
 - provide run-time information on software
 - allocation, use and de-allocation of resources, e.g. memory
- **useful when testing middleware**
- **interface to software under test**
 - run tests as if done manually
- **test scripts in a programmable language**
- **stores data, test inputs, scripts, etc. in repositories**
- **most often used to automate regression testing**
- **used to exercise software without a user interface**
- **facilitates testing by simulating an environment in which the test can be run**
 - simulators (where testing in real environment would be too costly / dangerous)
- **tools that support security testing**
 - virus checking, denial of service attacks
 - searching for vulnerabilities of the system
 - assist in the testing of “secure” websites
- **performance, load and stress tools**
 - drive application via user interface or test harness
 - simulates realistic load on the system, application, a database or environment
 - times for selected transactions via user interface

Exercise 6_1

Tool Pairs

Tool Definition

STF151030

Card 1_9

Tool Definition

STF151030

Card 1_10

Tool Definition

STF151030

Card 1_11

Tool Definition

STF151030

Card 1_12

Tool Definition

STF151030

Card 1_13

Tool Definition

STF151030

Card 1_14

Tool Definition

STF151030

Card 1_15

Tool Definition

STF151030

Card 1_16

**Monitoring
Tools**

**Monitoring
Tools**

**Monitoring
Tools**

**Monitoring
Tools**

**Monitoring
Tools**

**Monitoring
Tools**

**Monitoring
Tools**

**Monitoring
Tools**

Exercise 6_1

Tool Pairs

<div>Tool Type</div> <div>STF151030</div> <div>Card 1_17</div>	<div>Tool Type</div> <div>STF151030</div> <div>Card 2_17</div>
<div>Tool Type</div> <div>STF151030</div> <div>Card 3_17</div>	<div>Tool Type</div> <div>STF151030</div> <div>Card 4_17</div>
<div>Tool Type</div> <div>STF151030</div> <div>Card 5_17</div>	<div>Tool Type</div> <div>STF151030</div> <div>Card 6_17</div>
<div>Tool Type</div> <div>STF151030</div> <div>Card 7_17</div>	<div>Tool Type</div> <div>STF151030</div> <div>Card 8_17</div>

Tool Pairs

Exercise 6_1

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

- **continuously analyse, verify and report ...**
 - use of system resources
 - warnings of possible problems
- **store recorded information**
 - helps with traceability
- **often used by operations**

Exercise 6_1

Tool Pairs

**Tool
Definition**

STF151030

Card 1_17

**Tool
Definition**

STF151030

Card 2_17

**Tool
Definition**

STF151030

Card 3_17

**Tool
Definition**

STF151030

Card 4_17

**Tool
Definition**

STF151030

Card 5_17

**Tool
Definition**

STF151030

Card 6_17

**Tool
Definition**

STF151030

Card 7_17

**Tool
Definition**

STF151030

Card 8_17

Exercise Solution

Test Process Game

Solution 1_1

Test planning	
Assigning resources for the different activities defined	Reviewing the test basis
Defining test levels and entry and exit criteria	Test closure activities
Defining the level of detail, structure and templates for test documentation	Analysing lessons learned to determine changes needed for future releases and projects
Defining overall testing approach, including definition of test levels, entry & exit criteria	Checking which planned deliverables have been delivered
Determining the scope and risks, and identifying the objectives of testing	Closing incident reports or raising change records for any that remain open
Integrate testing into lifecycle activities: acquisition, supply, development, operation, maintenance	Documenting the acceptance of the system
Making decisions about what to test and what roles will perform the test activities	Finalising and archiving testware, test environment and test infrastructure for later use
Scheduling test analysis and design activities	Handover of testware to maintenance
Scheduling test implementation, execution and evaluation activities	Using the information gathered to improve test maturity
Select metrics to monitor & control preparation and execution, defect resolution and risk issues	Test implementation and execution
Verifying the mission of testing	Comparing actual results with expected results
Test control	Creating test suites from the test procedures for efficient execution
Change the test schedule due to availability of a test environment	Developing and prioritising test procedures, creating test data
Comparing actual progress against the plan	Developing, implementing and prioritising test cases
Making decisions based on feedback from monitoring and control activities	Executing test procedures manually or using test execution tools
Monitoring testing throughout the project	Logging the outcome of test execution and record identities and versions
Reporting the status of testing, including deviations from the plan	Optionally, preparing test harnesses and writing automated test scripts
Re-prioritise tests when an identified risk occurs (e.g. software delivered late)	Repeating test activities as a result of action taken for each discrepancy
Setting an entry criterion requiring fixes to have been retested by a developer	Reporting discrepancies as incidents and analysing them
Test analysis and design	Verifying that the test environment has been set up correctly
Creating bi-directional traceability between test basis and test cases.	Verifying and updating bi-directional traceability between the test basis and test cases
Designing and prioritising test cases	Evaluating exit criteria and reporting
Designing the test environment set-up, identifying infrastructure and tools	Assessing if more tests are needed or if the exit criteria should be changed
Evaluating testability of the test basis and test objects	Checking test logs against the exit criteria
Identifying and prioritising test conditions and required test data	Writing a test summary report for stakeholders
Identifying necessary test data to support test conditions and test cases	

Match Statements with Test Levels Solution

	Test Levels			
	Component Test	Integration Test	System Test	Acceptance Test
Objective of the test	5) Find defects in logic of the code, detailed boundary checking.	4) Find defects in interactions between components.	6) Find defects in end-to-end functionality.	7) Give confidence that the system is ready to deploy.
Who does the testing?	2) Developer or Buddy Developer.	3) Developer or Tester.	9) Independent test team (usually).	12) Users, customers, stakeholders.
Test Basis	1) Program specification, detailed design, data model.	8) Architectural design of how the code modules fit together.	11) Requirements, risks, use cases and functional design specification.	10) Requirements, risks and acceptance criteria.

Test Targets

Solution 2_2

Match Test Targets with Test Levels Solution

Test Type (Target)	Test Levels			
	Component Test	Integration Test	System Test	Acceptance Test
Functional	2) Testing of field level input for each unit or program.	12) Testing the transfer of information between modules and systems.	4) End to end testing of the whole system.	10) Testing to make sure the system does what it says it should do from a business perspective.
Non-Functional (Quality Characteristics)	8) Robustness, look for memory leaks, resource issues and maintainability.	3) Testing how fast information is transferred between components.	7) Testing reliability of the new system.	11) Testing that the system is usable from a business perspective.
Structural	9) Statement coverage, decision outcome coverage.	6) Module call coverage, interface coverage.	5) Menu option coverage, navigation coverage.	1) Coverage of business rules.

Reviews Exercise Sample Answers:

Advantages of an Informal Review
<ul style="list-style-type: none">• quick• better than nothing (?)• finds some defects• lots of people can be involved• usefulness depends on reviewers• can be beneficial when departments are located in different places
Advantages of a Walkthrough
<ul style="list-style-type: none">• gain understanding• learning experience• discussion is encouraged• finds defects (ambiguities and misunderstandings)• provides confidence about the authors knowledge• can be useful prior to individual checking of documents
Advantages of a Technical Review
<ul style="list-style-type: none">• focus on technical aspects – solving technical problems• discussion and decision making encouraged• finds defects• useful before sending document to other departments• use of checklists can increase effectiveness
Advantages of an Inspection
<ul style="list-style-type: none">• finds defects (usually lots)• can “chunk” or “sample”• very objective (based upon rule violations)• metrics obtained• process improvement built in• training is a must• entry and exit criteria are clear

Black Box

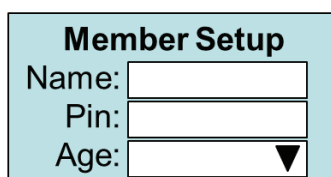
Solution 4_1

Analyse the test basis

The parts of the specification that need further explanation, are ambiguous or un-testable are highlighted in the specification below.

Specification

A new hard-drive recording system is required for households with up to 6 people. Each new household member will be set up with a unique name (4 – 8 characters), a pin number (4 digits) and an age category is chosen from a list (0-11, 12-17, 18+)



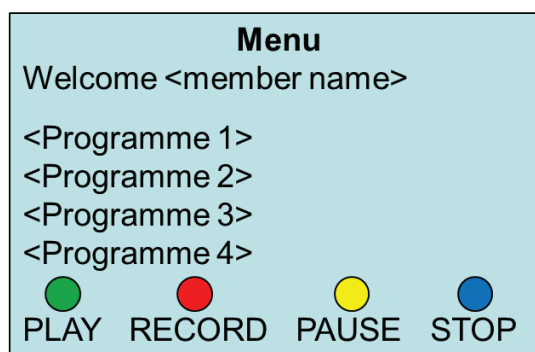
Member Setup

Name:

Pin:

Age: ▼

The age category of the member restricts the viewing for that particular member. 0-11 years can only watch "child" programmes, 12-17 years can watch "child" and "teenager" programmes only and 18+ years can watch all programmes (including "adult".) These restrictions limited to systems being used in Europe. Upon login the member is presented with a menu screen and programmes displayed (that relate to that member's age category) for selection.



Menu

Welcome <member name>

<Programme 1>

<Programme 2>

<Programme 3>

<Programme 4>

● ● ● ●

PLAY RECORD PAUSE STOP

Once a member is logged onto the system they can watch or record the allowable programme by using the buttons on the remote. When the member selects a programme they can either play or record by pressing the relevant button, a green or red light will display showing the status of the programme. Pressing the "stop" button will return the member to the menu screen. The programme can be paused (when playing or recording) by pressing the "pause" button. The relevant light will flash indicating that the programme has been paused. To resume play or record then the "pause" button is pressed again. When the programme is in a record/pause state it can be stopped by pressing the "stop" button.

Highlighted Text	Reason
up to	Ambiguous: does this include 6 or is the maximum 5?
unique name	Ambiguous: syntax not clear.
child, teenager, adult	Ambiguous: these are not standard ratings – how will we know which programmes fall into which category?
Europe	Ambiguous: European countries or European continent?
presented	Ambiguous: inconsistent description, is there a difference between 'presented' and 'displayed'?
green or red light	Ambiguous: specification must say which colour indicates what status.
status	Missing: no information about the possible 'status' values.
relevant light	Ambiguous: which is the relevant light?

Identify test conditions

Using Equivalence Partitioning and Boundary Value Analysis we can identify these test conditions:

	Valid Partition	Invalid Partition	Valid Boundary	Invalid Boundary
Member Name	4 – 8 chars and valid chars	< 4 chars	4 chars	3 chars
		> 4 chars	8 chars	9 chars
		Invalid chars		
Pin Number	4 digits	< 4 digits	4 digits (already covered in partition)	3 digits
		> 4 digits		5 digits
		Non-digit		
Age	Child	Other	There are no boundaries	
	Teenager			
	Adult			
Number of members	1 - 6	< 1	1 member	0 members
		> 6	6 members	7 members

Generate Test Cases

The table below shows test cases designed to cover some of the test conditions identified above:

Test Case	Description	Expected Results
EB1 Set up first member.	Precondition: no members on the system Member Name: John1 Pin No: 1234 Age: 22	Adult Member accepted
EB2 Set up a child.	Precondition: 3 members on the system Member name: Achie99 Pin No: 9999 Age: 6	Child member accepted
EB3 Name too long.	Precondition: 2 members on the system Member name: JohnSmith Pin No: 9998 Age: 45	Error: name too long
EB4 Attempt to set up more than 6 members.	Precondition: 6 members on the system Member name: Melanie Pin No: 2222 Age: 16	Error: too many members
....		

Black Box**Solution 4_1****Identify test conditions (continued)**

Using our rationalised decision table we can identify these test conditions:

Condition				
Age Category	----	0-11	12-17	18+
Europe	F	T	T	T
Action				
Child Programme	T	T	T	T
Teen Programme	T	F	T	T
Adult Programme	T	F	F	T
Tags	A	B	C	D

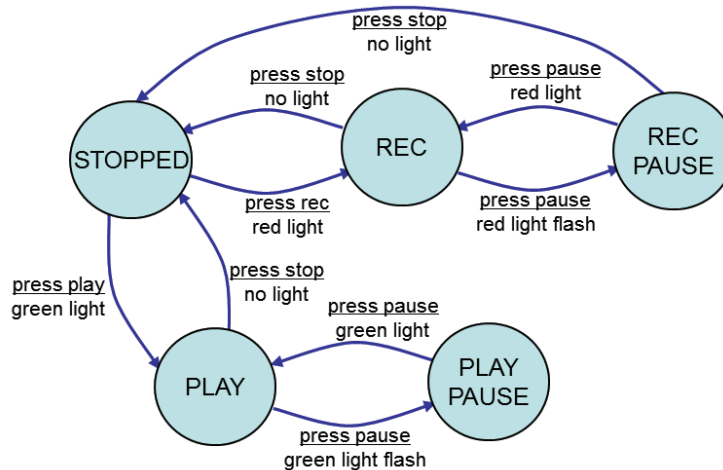
Generate test cases (continued)

The table below shows test cases designed to cover some of the test conditions identified above:

Test	Description	Expected Output	Tag
Test Case DT1	Age 6, non-european	member can watch and record any programme	A
Test Case DT2	Age 6, European	Member can watch and record child programmes only	B
Test Case DT3	Age 15, European	Member can watch and record child and teenager programmes only	C
Test Case DT4	Age 22, European	member can watch and record any programme	D

Identify test conditions (continued)

With State Transition Testing we identify test conditions using the state graph and state table below.



States \ Events	Press "stop"	Press "play"	Press "rec"	Press "pause"
STOPPED	---	PLAY	REC	---
PLAY	STOPPED	---	---	PLAY PAUSE
RECORD	STOPPED	---	---	REC PAUSE
PLAY PAUSE	---	---	---	PLAY
REC PAUSE	STOPPED	---	---	REC

Generate test cases (continued)

The table below shows test cases designed to cover some of the test conditions identified above:

Test Case	Description	Expected Result	Comment
ST1	Pre-requisite: Non-European member logged on, with programme already recorded Input: Presses play, play pause, play pause, stop	Member watches programme and is able to pause.	This is a typical transaction.
ST2	Pre-requisite: European, member aged 6 logged on. Input: press record, record pause, record pause, stop.	Child member in Europe can record a child programme.	This together with above test case will cover every state.
...			

Black Box**Solution 4_1****Produce test procedures**

Having generated the test conditions and test cases – put the test cases into a sequence order to produce a test procedure and prioritise this procedure.

This is an example of a test procedure with various steps, priorities and tests. This procedure is an example only.


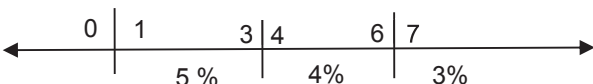
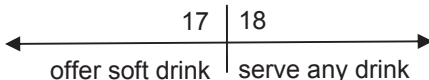
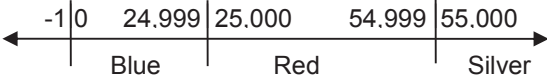
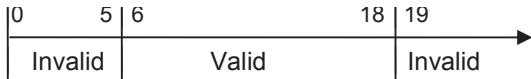
Step number	Description	Check
1	Precondition: 3 members on the system European country Programmes already recorded	
2	Run Test Case EB2	Child member accepted
3	Run Test Case DT1	That child can view child programmes only
4	Run Test Case ST2	That Child watches programme with a pause
5	Child logs off system	
6	Run Test Case EB1	Adult member accepted
7	Run Test Case DT4	Adult member can see all programmes to record
8	Run Test Case ST1	Adult member can record a programme with a pause
END		

It is worth mentioning that this procedure will be different to the one you create because it will depend on the level of detail that you want to include, your objectives for testing and also the priorities you have given to the test conditions and test cases.

Specification	Equivalence Partitions and Boundary Values
The cardholders name on a credit card processing payment screen can accept up to 32 characters. It cannot be left blank. The field can only accept the following set of characters: A to Z inclusive, a to z inclusive and space.	<p>Cardholder Name: number of characters</p> <p>Invalid Valid Invalid</p> <p>Valid: A-Z a-z space Invalid: any other characters</p>
A financial company decides to offer their customers a credit card based on annual salary. If the customer earns 35,000 euros or more, then they will be accepted for a gold card, if they earn 60,000 euros or more then they will receive platinum, otherwise they will be rejected for the offer.	<p>Salary</p> <p>Rejected Gold</p> <p>Question: what is the upper limit? Assumptions: gold card is invalid when platinum card issued and salary measured in whole euros ignoring the cents.</p>
In order to pass an exam you need to achieve 60% or more. However, if you achieve 80% or more you will be awarded a distinction.	<p>Exam grading</p> <p>Fail Pass Distinction</p> <p>Assumptions: minimum score is 0%; maximum score is 100%; below 60% is a fail.</p>
A thermostat display shows the temperature to the nearest degree Celsius from -10 to 30. When the temperature is below zero then the message 'Too cold' is displayed. When the temperature exceeds 20 degrees the message 'Too Hot' is displayed.	<p>Temperature (Degrees Celsius)</p> <p>Invalid Too cold ? Too hot Invalid</p> <p>Question: does it display anything between 0 and 20 degrees?</p>

Solution 4_2

Equivalence Partitioning & Boundary Value Analysis

Specification	Equivalence Partitions and Boundary Values
Phone calls are charged according to the time of day the call is made. If a call is made from 08:00 then it will be charged at 1 euro per minute. If it is made from 13:00 calls are charged at 1.50 euro per minute; calls from 17:00 to 22:59 inclusive are charged at 2 euro per minute. All other times will incur no charge.	<p style="text-align: center;">Phone Charges</p>  <p style="text-align: center;">Assumption: charging tariff will be made on whole minutes.</p>
An on-line system charges the following shipping costs. For orders of up to 3 books the shipping cost is 5% of the total cost. Orders of 4 to 6 books inclusive are charged a 4% shipping cost and orders of 7 or more books are charged a 3% shipping cost.	<p style="text-align: center;">Number of Books</p>  <p>Question: is there an upper boundary for 3% shipping cost? Assumptions: ordering zero books is invalid, ordering a negative number of books is invalid.</p>
A barman will serve you alcohol if you are at least 18 years of age. Anyone under this age will be offered a soft drink only.	<p style="text-align: center;">Age of customer</p>  <p>Assumption: only whole years are used. Question: are other boundaries needed? (e.g. what is youngest age you can enter the bar?).</p>
The types of loyalty card available will be determined by the points accrued on the reward scheme. All customers start on a blue card. After accruing 25,000 points they will receive a red card and after a further 30,000 points they will be eligible for a silver card.	<p style="text-align: center;">Loyalty Points</p>  <p>Question: what is the upper limit?</p>
An application requires a telephone number comprising only digits to be entered. A minimum of 6 digits and a maximum of 18 digits are accepted.	<p style="text-align: center;">Telephone number (number of digits)</p>  <p style="text-align: center;">Invalid: non-digits</p>

Decision Table

Solution 4_3

Step 1: Decision table

Condition								
Valid number?	F	F	F	F	T	T	T	T
Chargeable	F	F	T	T	F	F	T	T
Account in credit	F	T	F	T	F	T	F	T
Action								
Call connected	F	F	F	F	T	T	F	T
Account debited	F	F	F	F	F	F	F	T
Tags:	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>	<i>H</i>

Step 2: Can this decision table be rationalised?

Yes! If the number is not valid, it doesn't make sense to consider anything else, so we can eliminate columns with Tags B, C and D. Also, if the call is free then account status doesn't affect the outcome; so, we can eliminate one of the columns tagged E and F.

Remember rationalisation is based upon assumptions, which may be wrong so it is important to check the assumption is correct.

Condition				
Valid number?	F	T	T	T
Chargeable	-	F	T	T
Account in credit	-	-	F	T
Action				
Call connected	F	T	F	T
Account debited	F	F	F	T
Tags:	<i>A</i>	<i>F</i>	<i>G</i>	<i>H</i>

'-' don't care

Test Case 1 – Tag H

Input: valid number, chargeable and account in credit

Output: call connected, account debited

Test Case 2 – Tag G

Input: valid number, chargeable but account not in credit

Output: call not connected, account not debited

State Transition

Solution 4_4

a) A test case for a **typical transaction**.

S1 - tea - S2 - milk – S4 – dispensed – S2 - vend – S3 – take drink – S1 –
purchasing cup of tea, with milk but no sugar

b) A test case **that will go through every state**.

S1 - tea - S2 - milk – S4 - dispensed – S2 – sugar – S5 – dispensed – S2 –
vend – S3 – take drink – S1

c) A test case **that will exercise every transition**.

S1 - tea - S2 - milk – S4 - dispensed – S2 – sugar – S5 – dispensed – S2 –
vend – S3 – take drink – S1 – coffee – S2 – clear – S1

This is not the only test that would exercise every transition.

d) Complete the following State Table and identify any invalid transitions.

	tea	coffee	clear	sugar	milk	vend	dispensed	take drink
S1 selection	S2	S2	--	--	--	--	--	--
S2 extras	--	--	S1	S5	S4	S3	--	--
S3 serve drink	--	--	--	--	--	--	--	S1
S4 milk	--	--	--	--	--	--	S2	--
S5 sugar	--	--	--	--	--	--	S2	--

Invalid transitions are denoted by "--" in the table above, such as take your drink whilst in the milk state.

1. For the following:

IF you buy a cheap-day return ticket **THEN**
 catch a train after 9.30am
ELSE
 catch any train
ENDIF
 Buy a coffee
 Read the newspaper
 Enjoy the train journey

a) What is the minimum number of test cases that are required to achieve 100% Statement Coverage?	2
b) What is the minimum number of test cases that are required to achieve 100% Decision Coverage?	2
c) How much decision coverage have I achieved if I buy a “cheap-day-return” ticket only?	50% (1 out of 2)

2. For the following pseudo code:

Read A
 Read B
IF B = A **THEN**
 Print “they are the same”
ELSE
 Print “they are different”
ENDIF
 Print “End of processing”

Generate a minimum set of tests (values of A and B) that achieve 100% statement and 100% decision coverage
Test Case 1: A = 5, B = 5 – expected output “they are the same”
Test Case 2: A = 5, B = 4 – expected output “they are different”

3. For the following :

IF the vending machine is not working **THEN**
 call repair centre to fix
ELSE
 Insert money
 WHILE there is not enough money
 Display message “insert money”
 Insert more money
 ENDWHILE
 Select a drink
 Wait for drink to be dispensed
 Enjoy the drink!
 Collect any change
ENDIF

c) Calculate the minimum number of tests required to achieve 100% statement coverage	2
d) Calculate the minimum number of tests required to achieve 100% decision coverage	2

Solution 4_5

Structural Techniques

4. Given the following pseudo code:

```

Read P
Read Q
IF P+Q > 100 THEN
    Print "Large"
ELSE
    IF P+Q > 50 THEN
        Print "Medium"
    ENDIF
ENDIF

```

a) What is the minimum number of test cases that are required to achieve 100% Statement Coverage:	2
b) What is the minimum number of test cases that are required to achieve 100% Decision Coverage	3
c) Provide values for P and Q that will achieve 100% Statement Coverage	Test Case 1: P = 50, Q = 60 Test Case 2: P = 50, Q = 50
d) Will these same values achieve 100% Decision Coverage	No, a further test is needed, e.g. P = 20, Q = 20

5. For the following pseudo code:

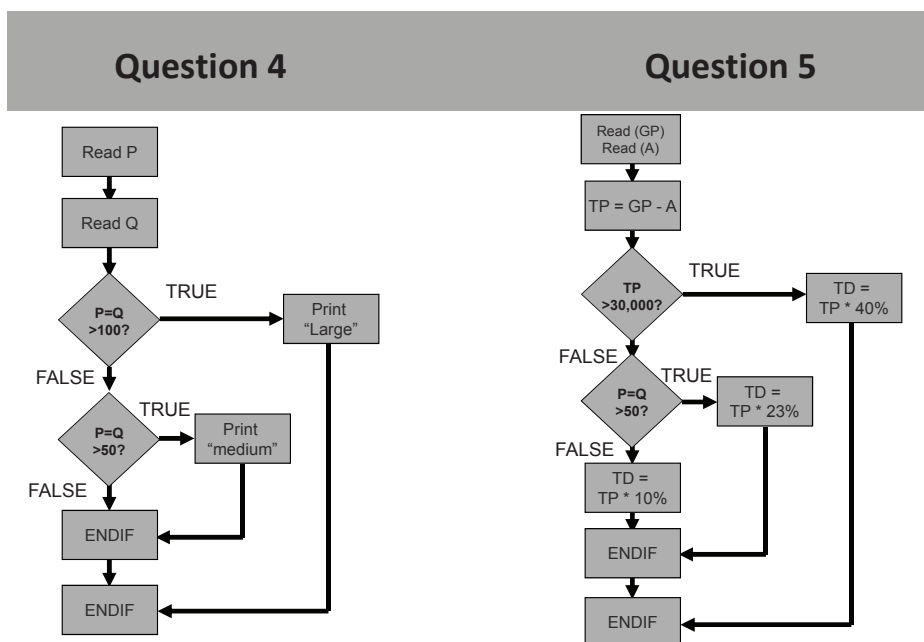
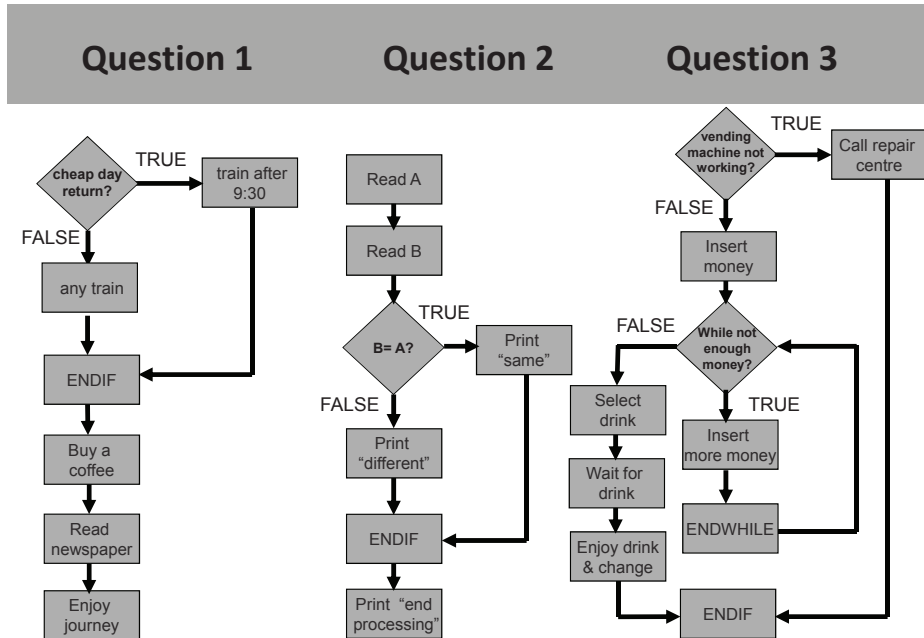
```

Read (Gross Pay)
Read (Allowances)
Taxable Pay = Gross Pay - Allowances
IF Taxable Pay > 30,000 THEN
    Tax Due = Taxable Pay * 40%
ELSE
    IF Taxable Pay > 15,000 THEN
        Tax Due = Taxable Pay * 23%
    ELSE
        Tax Due = Taxable Pay * 10%
    ENDIF
ENDIF

```

a) What is the minimum number of test cases that are required to achieve 100% Statement Coverage:	3
b) What values are required for Gross Pay and Allowances to achieve 100% Decision Coverage? Test Case 1: Gross Pay = 50,000, Allowances = 15,000 (Taxable Pay = 35,000) Test Case 2: Gross Pay = 50,000, Allowances = 22,000, (Taxable Pay = 28,000) Test Case 3: Gross Pay = 50,000, Allowances = 40,000, (Taxable Pay = 10,000)	

Structural Techniques Solution - Diagrams



Incident

Solution 5_1

Incident Report

Incident ID: 24	Test ID: BL246	Software Version: Puzzle 2.0
Incident Description. Include actual and expected results: Moving a number from the top right cell results in the number overwriting another number (in the position last moved to from this game or the previous game (or the 2 nd cell on the top row if no moves have been made). A second click on the top right cell makes the number just moved disappear.		Severity: 1 = Critical or Testing Stopped 2 = Major or Test Impeded 3 = Minor 4 = Cosmetic / Query / Comment Priority: High Medium Low Attachments: <ul style="list-style-type: none"> • Screen shot. • Steps taken.
Investigation results: Name: Testing Tim Date: 12/4/XX		Cause Code:
Resolution: Name: Date:		Status: Open In development In testing Closed
Closed By:		Date:

Tool Type	Tool Description
Configuration Management Tools	<ul style="list-style-type: none"> • provides valuable infra-structure for testing to succeed <ul style="list-style-type: none"> – not strictly testing tools • what can they do? <ul style="list-style-type: none"> – stores information about versions/builds of software – traceability between testware and software
Coverage Measurement Tools	<ul style="list-style-type: none"> • objective measure of test coverage <ul style="list-style-type: none"> – tool reports what has and has not been covered by those tests, line by line and summary statistics • different types of coverage <ul style="list-style-type: none"> – statement, branch, condition, LCSAJ, etc.
Dynamic Analysis Tools	<ul style="list-style-type: none"> • helps find defects that are only evident when the software is run <ul style="list-style-type: none"> – provide run-time information on software – allocation, use and de-allocation of resources, e.g. memory • useful when testing middleware
Incident Management Tools	<ul style="list-style-type: none"> • also known as defect tracking/bug logging tools <ul style="list-style-type: none"> – commercial or in-house built • stores, manages & monitors incident reports <ul style="list-style-type: none"> – ability to log information concerning the incident • provides support for various analysis
Modelling Tools	<ul style="list-style-type: none"> • validate models of the software <ul style="list-style-type: none"> – finding defects in data, state and/or object models • aids test case generation <ul style="list-style-type: none"> – where test cases are based on the model
Monitoring Tools	<ul style="list-style-type: none"> • continuously analyse, verify and report ... <ul style="list-style-type: none"> – use of system resources – warnings of possible problems • store recorded information <ul style="list-style-type: none"> – helps with traceability • often used by operations
Performance Testing Tools	<ul style="list-style-type: none"> • performance, load and stress tools <ul style="list-style-type: none"> – drive application via user interface or test harness – simulates realistic load on the system, application, a database or environment – times for selected transactions via user interface
Requirement Management Tools	<ul style="list-style-type: none"> • automated support for verification and validation of requirements models <ul style="list-style-type: none"> – stores requirement statements – checks for consistency, helps prioritise • traceability to design, code, tests

Solution 6_1

Tool Pairs

Tool Type	Tool Description
Review Process Support Tools	<ul style="list-style-type: none"> • mostly “in-house” built <ul style="list-style-type: none"> – spreadsheets, databases and templates • what can they do? <ul style="list-style-type: none"> – stores information about the review process – comments, defects, effort, changes, metrics
Security Tools	<ul style="list-style-type: none"> • tools that support security testing <ul style="list-style-type: none"> – virus checking, denial of service attacks – searching for vulnerabilities of the system – assist in the testing of “secure” websites
Static Analysis Tools	<ul style="list-style-type: none"> • provides information about the quality of software <ul style="list-style-type: none"> – code is examined, not executed • supports developers, testers and QA teams <ul style="list-style-type: none"> – finds defects before test execution – saves time & money
Test Comparators	<ul style="list-style-type: none"> • detect differences between actual test results and expected results <ul style="list-style-type: none"> – screens, characters, bitmaps – masking and filtering
Test Data Preparation Tools	<ul style="list-style-type: none"> • data manipulation <ul style="list-style-type: none"> – selected from existing databases or files – created according to some rules – edited from other sources – safeguards “data protection act” if using live-data
Test Design Tools	<ul style="list-style-type: none"> • generate test inputs <ul style="list-style-type: none"> – from a formal specification – from a CASE repository (design model) – from code (e.g. code not covered yet) • generate “limited” expected outcomes
Test Execution Tools	<ul style="list-style-type: none"> • interface to software under test <ul style="list-style-type: none"> – run tests as if done manually • test scripts in a programmable language • stores data, test inputs, scripts, etc. in repositories • most often used to automate regression testing
Test Harnesses/Unit Test Framework	<ul style="list-style-type: none"> • used to exercise software without a user interface • facilitates testing by simulating an environment in which the test can be run <ul style="list-style-type: none"> – simulators (where testing in real environment would be too costly / dangerous)
Test Management Tools	<ul style="list-style-type: none"> • management of testware (test plans, specifications, etc.) • interfaces to other tools <ul style="list-style-type: none"> – e.g. test execution, incident logging and CM • traceability of test objects to requirement specifications

Practice Exam

Practice Exam 1

1. **Which of the following could be a root cause of a defect?**
 - a) a report displays the wrong totals
 - b) the calculation of square roots was 10% too low
 - c) not enough time was given to review the requirement specification
 - d) the error message produced was incorrect
2. **Which of the following could be caused by a defect in software?**
 - 1) damage to a company's value on the stock exchange
 - 2) a person didn't follow instructions and poured the wrong chemical into a storage tank
 - 3) injury by a mis-behaving industrial robot
 - 4) a mobile phone intermittently re-boots itself
 - a) 1, 2 and 3
 - b) 1, 2 and 4
 - c) 1, 3 and 4
 - d) 2, 3 and 4
3. **A defect is defined as:**
 - a) actual deviation of the component or system from its expected delivery, service or result
 - b) a flaw in a component or system that can cause the component or system to fail to perform its required function
 - c) a malfunction by the component or system resulting from the software being run
 - d) a mistake made by the developer that produces an incorrect result
4. **Match the following activities with the sample tasks given below**
 - q) Test Planning and Control
 - r) Test Analysis and Design
 - s) Test Implementation and Execution
 - t) Evaluating Exit Criteria and Reporting
 - u) Test Closure Activities
 - v) Assess whether more tests are needed
 - w) Decide what metrics to monitor
 - x) Create test inputs and expected results
 - y) Set up the test environment, compare actual with expected results
 - z) Handover of testware to maintenance
 - a) q=w; r=v; s=y; t=x; u=z
 - b) q=w; r=x; s=y; t=v; u=z
 - c) q=x; r=w; s=y; t=v; u=z
 - d) q=x; r=w; s=v; t=z; u=y

- 5. The decision about how much testing is enough should be based on:**
- a) product and project risks
 - b) project staffing, project risks and project constraints
 - c) product and project risks and project constraints
 - d) project constraints, particularly time and deadlines
- 6. If the same tests are repeated over and over again,**
- a) more and more defects will be found
 - b) different defects are likely to be found each time they are run
 - c) eventually the tests will no longer find any new defects
 - d) eventually the tests will find all of the defects
- 7. Which of the following is true:**
- a) testing can find defects and can prove there are no defects
 - b) testing can show that there are defects but cannot prove that there are no defects
 - c) testing can prove there are no defects but cannot show that there are defects
 - d) testing can prove that all defects have been found
- 8. Which of the following are the BEST reasons for testing being necessary?**
- 1. testing is part of the development lifecycle
 - 2. testing assesses the quality of the software
 - 3. there are defects in the software
 - 4. testing finds out about the reliability of the software
- a) 2 and 3
 - b) 1, 2 and 4
 - c) 1 and 3
 - d) 2, 3 and 4
- 9. Which of the following have the most influence on the success of testing?**
- 1. the psychological profile of the managers
 - 2. a balance of self-testing and independence
 - 3. clear objectives for testing
 - 4. the tools and environment used by developers
 - 5. courteous communication and feedback about defects
- a) 2, 3 and 4
 - b) 1, 2 and 3
 - c) 1, 4 and 5
 - d) 2, 3 and 5
- 10. Which of the following statements is true?**
- a) Dynamic testing tries to identify defects in the software
 - b) Dynamic testing tries to generate failures of the system
 - c) Dynamic testing tries to fix defects in the software
 - d) Dynamic testing tries to prove that there are no defects in the software

Practice Exam Answers

Practice Exam 1 Answers

1.

Which of the following displays the wrong total cause of a defect?

- a) the calculation of square roots was 10% too low
- b) not enough time was given to develop the requirement specification
- c) the time was given to develop the requirement specification
- d) the time was given to develop the requirement specification

2.

Which of the following could be caused by a defect in software?

- a) damage to a company's value on the stock exchange
- b) a person didn't follow instructions and poured the wrong chemical into a storage tank
- c) injury by a mis-behaving industrial robot
- d) a mobile phone intermittently re-boots itself

- a) 1, 2 and 3
- b) 1, 2 and 4
- c) 1, 3 and 4
- d) 2, 3 and 4

3. A defect is defined as:

- a) actual deviation of the component or system from its expected delivery, service or result
- b) a flaw in a component or system that can cause the component or system to fail to perform its required function
- c) a malfunction by the component or system resulting from the software being run
- d) a mistake made by the developer that produces an incorrect result

4.

Match the following principles with the sample tasks given below

- r) Test Analysis and Design
- s) Test Implementation and Execution
- t) Evaluating Exit Criteria and Reporting
- u) Test Closure Activities
- v) Assess whether more tests are needed
- w) Decide what metrics to monitor
- x) Create test inputs and expected results
- y) Set up the test environment, compare actual with expected results
- z) Handover of testware to maintenance

- a) q=w; r=v; s=y; t=x; u=z
- b) q=w; r=x; s=y; t=v; u=z
- c) q=x; r=w; s=y; t=v; u=z
- d) q=x; r=w; s=v; t=z; u=y

5. The decision about how much testing is enough should be based on:
- product and project risks
 - project staffing, project risks and project constraints
 - product and project risks and project constraints
 - project constraints, particularly time and deadlines
6. If the same tests are repeated over and over again,
- more and more defects will be found
 - different defects are likely to be found each time they are run
 - eventually the tests will no longer find any new defects
 - eventually the tests will find all of the defects
7. Which of the following is true:
- testing can find defects and can prove there are no defects
 - testing can show that there are defects but cannot prove that there are no defects
 - testing can prove there are no defects but cannot show that there are defects
 - testing can prove that all defects have been found
8. Which of the following are the BEST reasons for testing being necessary?
- testing is part of the development lifecycle
 - testing assesses the quality of the software
 - there are defects in the software
 - testing finds out about the reliability of the software
- 1, 2 and 3
 - 1, 2 and 4
 - 1, 3 and 4
 - 2, 3 and 4
9. Which of the following have the most influence on the success of testing?
- the psychological profile of the managers
 - a balance of self-testing and independence
 - clear objectives for testing
 - the tools and environment used by developers
 - courteous communication and feedback about defects
- 2, 3 and 4
 - 1, 2 and 3
 - 1, 4 and 5
 - 2, 3 and 5
10. Which of the following statements is true?
- Dynamic testing tries to identify defects in the software
 - Dynamic testing tries to generate failures of the system
 - Dynamic testing tries to fix defects in the software
 - Dynamic testing tries to prove that there are no defects in the software

Practice Exam 2

1. **Which statement about the characteristics of testing in any lifecycle is FALSE?**
 - a) for every development activity there should be a corresponding test activity
 - b) test design should be produced as soon as the software is available
 - c) testers should be involved in reviewing documents as soon as drafts are available
 - d) each test level has a test objective specific to that level
2. **Which of the following statements is true?**
 - a) Fixing defects is the most important thing for the users
 - b) Fixing defects doesn't help if the system is unusable
 - c) Fixing defects is the responsibility of the test team
 - d) Fixing defects is helpful to users even if it doesn't fulfil their expectations
3. **Which of the following test types (targets of testing) typically apply mostly to system testing?**
 - 1) Testing of function
 - 2) Testing of software product characteristics
 - 3) Testing of software structure / architecture
 - 4) Testing related to changes to an existing system
 - a) 2, 3 and 4
 - b) 1, 2 and 4
 - c) 1, 3 and 4
 - d) 1, 2 and 3
4. **Which of the following statements about regression testing is true?**
 - a) Regression testing is the only type of testing done in maintenance testing
 - b) Regression testing can include functional, non-functional and structural testing
 - c) Regression testing is not done at component test level
 - d) Regression testing is best done manually
5. **Which of the following statements about maintenance testing is true?**
 - a) Maintenance testing is only concerned with the testing of changes or enhancements
 - b) Maintenance testing should be more thorough than development testing
 - c) Maintenance testing is done when the system is transferred to another platform or environment
 - d) Maintenance testing is NOT needed when a decision is made that the system will be taken out of use, i.e. retired
6. **What is impact analysis within maintenance testing?**
 - a) Assessing the impact of a defect on the system's behaviour
 - b) Assessing the impact of a requested enhancement on business processes
 - c) Assessing the impact of a change to determine how much regression testing to do
 - d) Assessing the impact of a new person joining the test team

7. Match the following terms with the following definitions

- q) Test planning and control
- r) Test analysis and design
- s) Test implementation and execution
- t) Evaluating exit criteria and reporting
- u) Test closure activities

- v) Check test logs against planned criteria, assess whether more tests are needed
- w) Verify test mission and objectives, specify test activities, compare progress to plan
- x) Review test basis, transform requirements into test conditions and test cases
- y) Transform test cases into test procedures, create testware, set up environment, run tests
- z) Check deliverables, archive testware, analyse lessons learned

- a) q=x; r=w; s=y; t=v; u=z
- b) q=w; r=v; s=y; t=x; u=z
- c) q=w; r=x; s=y; t=v; u=z
- d) q=x; r=w; s=v; t=z; u=y

8. Why is independence important in testing?

- a) An independent tester may be more effective at finding defects missed by the person who wrote the software
- b) An independent tester will find defects more quickly than the person who wrote the software
- c) An independent tester will be more focused on showing how the software works than the person who wrote the software
- d) An independent tester will be more efficient because they are less familiar with the software than the person who wrote it

9. What are key characteristics of iterative-incremental software development model?

- a) Test activities done throughout the increment and regression testing for previous increment are needed
- b) Testing is only performed at the end of each increment
- c) Test activities will be done for each increment, no need for regression testing
- d) Test execution tools must be used in this type of software development model

10. Consider the following statements:

- 1 alpha testing is performed by potential customers for the product
- 2 alpha testing must be performed before beta testing
- 3 alpha testing is performed at customer sites
- 4 alpha testing should be performed when the software is stable
- 5 alpha testing is performed by customers or representatives at an in-house site

- a) 1 - 3 are true; 4 & 5 are false
- b) 1, 4 & 5 are true; 2 & 3 are false
- c) 1 & 4 are true; 2, 3 & 5 are false
- d) 4 & 5 are true, 1 - 3 are false

Practice Exam 2 Answers

1. **Which statement about the characteristics of testing in any lifecycle is FALSE?**
 - a) for every development activity there should be a corresponding test activity
 - b) test design should be produced as soon as the software is available
 - c) testers should be involved in reviewing documents as soon as drafts are available
 - d) each test level has a test objective specific to that level
2. **Which of the following statements is true?**
 - a) Fixing defects is the most important thing for the users
 - b) Fixing defects doesn't help if the system is unusable
 - c) Fixing defects is the responsibility of the test team
 - d) Fixing defects is helpful to users even if it doesn't fulfil their expectations
3. **Which of the following test types (targets of testing) typically apply mostly to system testing?**
 - 1) Testing of function
 - 2) Testing of software product characteristics
 - 3) Testing of software structure / architecture
 - 4) Testing related to changes to an existing system
 - a) 2, 3 and 4
 - b) 1, 2 and 4
 - c) 1, 3 and 4
 - d) 1, 2 and 3
4. **Which of the following statements about regression testing is true?**
 - a) Regression testing is the only type of testing done in maintenance testing
 - b) Regression testing can include functional, non-functional and structural testing
 - c) Regression testing is not done at component test level
 - d) Regression testing is best done manually
5. **Which of the following statements about maintenance testing is true?**
 - a) Maintenance testing is only concerned with the testing of changes or enhancements
 - b) Maintenance testing should be more thorough than development testing
 - c) Maintenance testing is done when the system is transferred to another platform or environment
 - d) Maintenance testing is NOT needed when a decision is made that the system will be taken out of use, i.e. retired
6. **What is impact analysis within maintenance testing?**
 - a) Assessing the impact of a defect on the system's behaviour
 - b) Assessing the impact of a requested enhancement on business processes
 - c) Assessing the impact of a change to determine how much regression testing to do
 - d) Assessing the impact of a new person joining the test team

7. Match the following terms with the following definitions

- q) Test planning and control
- r) Test analysis and design
- s) Test implementation and execution
- t) Evaluating exit criteria and reporting
- u) Test closure activities

- v) Check test logs against planned criteria, assess whether more tests are needed
- w) Verify test mission and objectives, specify test activities, compare progress to plan
- x) Review test basis, transform requirements into test conditions and test cases
- y) Transform test cases into test procedures, create testware, set up environment, run tests
- z) Check deliverables, archive testware, analyse lessons learned

- a) q=x; r=w; s=y; t=v; u=z
- b) q=w; r=v; s=y; t=x; u=z
- c) q=w; r=x; s=y; t=v; u=z
- d) q=x; r=w; s=v; t=z; u=y

8. Why is independence important in testing?

- a) An independent tester may be more effective at finding defects missed by the person who wrote the software
- b) An independent tester will find defects more quickly than the person who wrote the software
- c) An independent tester will be more focused on showing how the software works than the person who wrote the software
- d) An independent tester will be more efficient because they are less familiar with the software than the person who wrote it

9. What are key characteristics of iterative-incremental software development model?

- a) Test activities done throughout the increment and regression testing for previous increment are needed
- b) Testing is only performed at the end of each increment
- c) Test activities will be done for each increment, no need for regression testing
- d) Test execution tools must be used in this type of software development model

10. Consider the following statements:

- 1 alpha testing is performed by potential customers for the product
- 2 alpha testing must be performed before beta testing
- 3 alpha testing is performed at customer sites
- 4 alpha testing should be performed when the software is stable
- 5 alpha testing is performed by customers or representatives at an in-house site

- a) 1 - 3 are true; 4 & 5 are false
- b) 1, 4 & 5 are true; 2 & 3 are false
- c) 1 & 4 are true; 2, 3 & 5 are false
- d) 4 & 5 are true, 1 – 3 are false

Practice Exam 3

1. Given the following decision table, what is the expected result for the test case listed below?

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
< 10 kg?	True	True	False	False
< €10?	True	False	True	False
Actions				
Must pay in cash only	True	False	True	False
Free delivery	False	False	False	True

What is the expected result for the following test case?

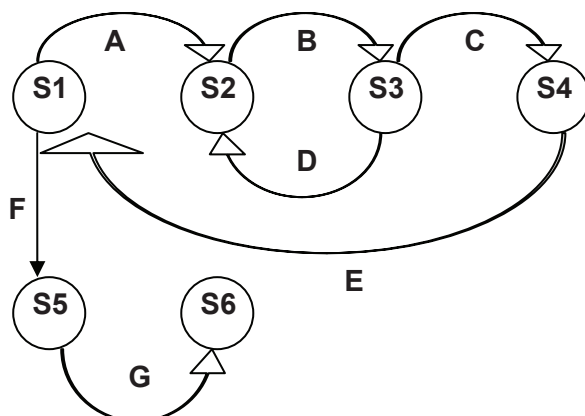
TC1: Purchase of a box of paper weighing 2 kg, for €9.95

- a) Don't need to pay in cash, and no free delivery
- b) Need to pay in cash, and no free delivery
- c) Don't need to pay in cash, and free delivery
- d) Need to pay in cash, and free delivery

2. Under what circumstances should you plan to test a purchased (3rd party) package?

- a) it should not be necessary if the package is purchased from a reputable supplier
- b) if you suspect that the supplier's testing may be inadequate
- c) if the package is critical to your own system
- d) if the package is very large

3. Given the following state transition diagram



Which of the following series of state transitions below will cover all transitions?

- a) A, B, C, D, E, F, G
- b) A, B, D, B, C, E, F, G
- c) A, B, C, D, E,
- d) A, B, D, B, C, E, A, B, C, E, F

4. Given the following state table, which is an INVALID transition?

	A	B	C	D	E
S1	S2	-	-	-	-
S2	-	S3	-	-	-
S3	-	-	S4	S2	-
S4	-	-	-	-	S1

- a) event E while in State 3
- b) event E while in State 4
- c) event D while in State 3
- d) event B while in State 2

5. Maintenance testing is:

- a) done on an existing operational system, triggered by modifications, migration or retirement of the system
- b) determined by the size of a change to an existing operational system
- c) the updating of the regression test pack that is run when an operational system is modified, migrated or retired
- d) done on a developing system before it is released to operation

6. Operational acceptance testing is

- a) testing that the system is operational and is acceptable to business users
- b) testing against the contract to ensure that all requirements have been met
- c) done by representatives from the system's operational profile, and includes assessment of market value
- d) done by system administrators, and includes user management, disaster recovery and checking for security vulnerabilities

7. What are the characteristics of good testing within a life cycle model?

- 1) test analysis and design should begin during the corresponding development activity for each level
 - 2) testers need not be involved in reviewing development documents, only test documents
 - 3) test objectives should be consistent across all test levels
 - 4) each development activity should have a corresponding test activity
 - 5) test analysis is based on development documents
- a) 3, 4 and 5
 - b) 2, 3 and 5
 - c) 1, 2 and 4
 - d) 1, 4 and 5

8. Which type of tool would be most likely to be used by a developer during component testing?

- a) Test harness or unit test framework
- b) Performance testing tool
- c) Monitoring tool
- d) Test management tool

9. Which of the following statements about static analysis tools are true?

- 1) Static analysis tools find defects rather than failures
 - 2) Static analysis tools are typically used by developers
 - 3) Static analysis tools measure code coverage
 - 4) Static analysis tools may produce a lot of warning messages
- a) 2, 3 and 4
 - b) 1, 2 and 3
 - c) 1, 2 and 4
 - d) 1, 3 and 4

10. A bank sets interest rates on an account depending on the amount in the account. From €0 to €500 inclusive the interest rate is 2.5%. From €500.01 to €1,000 it is 2.6%, from €1,000.01 to €1,500 it is 2.7%, from €1,500.01 to €2,000 it is 2.8%.

Which savings amounts below are all in DIFFERENT equivalence classes?

- a) €25, €150, €250, €499
- b) €525, €595, €1595, €1995
- c) €25, €250, €750, 1550
- d) €250, €750, €1250, €1995

Practice Exam 3 Answers

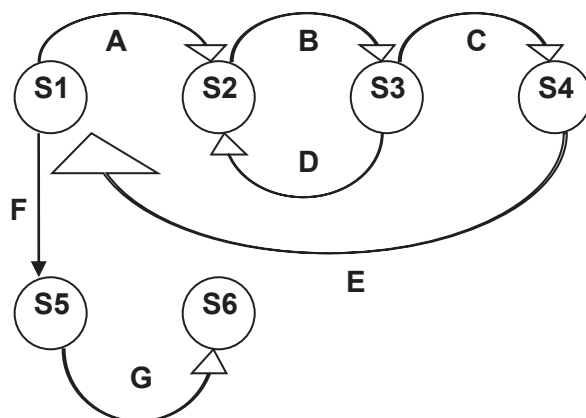
1. Given the following decision table, what is the expected result for the test case listed below?

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
< 10 kg?	True	True	False	False
< €10?	True	False	True	False
Actions				
Must pay in cash only	True	False	True	False
Free delivery	False	False	False	True

What is the expected result for the following test case?

TC1: Purchase of a box of paper weighing 2 kg, for €9.95

- a) Don't need to pay in cash, and no free delivery
 - b) Need to pay in cash, and no free delivery**
 - c) Don't need to pay in cash, and free delivery
 - d) Need to pay in cash, and free delivery
2. Under what circumstances should you plan to test a purchased (3rd party) package?
- a) it should not be necessary if the package is purchased from a reputable supplier
 - b) if you suspect that the supplier's testing may be inadequate
 - c) if the package is critical to your own system**
 - d) if the package is very large
3. Given the following state transition diagram



Which of the following series of state transitions below will cover all transitions?

- a) A, B, C, D, E, F, G
- b) A, B, D, B, C, E, F, G**
- c) A, B, C, D, E,
- d) A, B, D, B, C, E, A, B, C, E, F

4. Given the following state table, which is an INVALID transition?

	A	B	C	D	E
S1	S2	-	-	-	-
S2	-	S3	-	-	-
S3	-	-	S4	S2	-
S4	-	-	-	-	S1

- a) event E while in State 3
- b) event E while in State 4
- c) event D while in State 3
- d) event B while in State 2

5. Maintenance testing is:

- a) done on an existing operational system, triggered by modifications, migration or retirement of the system
- b) determined by the size of a change to an existing operational system
- c) the updating of the regression test pack that is run when an operational system is modified, migrated or retired
- d) done on a developing system before it is released to operation

6. Operational acceptance testing is

- a) testing that the system is operational and is acceptable to business users
- b) testing against the contract to ensure that all requirements have been met
- c) done by representatives from the system's operational profile, and includes assessment of market value
- d) done by system administrators, and includes user management, disaster recovery and checking for security vulnerabilities

7. What are the characteristics of good testing within a life cycle model?

- 1) test analysis and design should begin during the corresponding development activity for each level
 - 2) testers need not be involved in reviewing development documents, only test documents
 - 3) test objectives should be consistent across all test levels
 - 4) each development activity should have a corresponding test activity
 - 5) test analysis is based on development documents
- a) 3, 4 and 5
 - b) 2, 3 and 5
 - c) 1, 2 and 4
 - d) 1, 4 and 5

8. Which type of tool would be most likely to be used by a developer during component testing?

- a) Test harness or unit test framework
- b) Performance testing tool
- c) Monitoring tool
- d) Test management tool

9. Which of the following statements about static analysis tools are true?

- 1) Static analysis tools find defects rather than failures
 - 2) Static analysis tools are typically used by developers
 - 3) Static analysis tools measure code coverage
 - 4) Static analysis tools may produce a lot of warning messages
- a) 2, 3 and 4
 - b) 1, 2 and 3
 - c) 1, 2 and 4
 - d) 1, 3 and 4

10. A bank sets interest rates on an account depending on the amount in the account. From €0 to €500 inclusive the interest rate is 2.5%. From €500.01 to €1,000 it is 2.6%, from €1,000.01 to €1,500 it is 2.7%, from €1,500.01 to €2,000 it is 2.8%.

Which savings amounts below are all in DIFFERENT equivalence classes?

- a) €25, €150, €1550, €1999
- b) €525, €595, €1595, €1995
- c) €25, €250, €750, 1550
- d) €250, €750, €1250, €1995

Practice Exam 4

1. What do walkthroughs, technical reviews and inspections have in common?

- I. They have defect finding as an objective.
- II. Pre-meeting preparation is required.
- III. They can be performed as a “peer review”.
- IV. The meeting is led by a trained moderator.

- a) I, II and III
- b) III and IV
- c) I and III
- d) II, III and IV

2. User Acceptance Testing is most appropriately performed by:

- a) a test team of software developers
- b) business users
- c) the project manager
- d) the support group or support staff

3. Given the following test types, which of the following is an objective for that type?

- v) functional
 - w) non-functional
 - x) structural
 - y) change-related
 - q) defect fixes and unexpected side-effects
 - r) assesses the thoroughness of the testing
 - s) how the system works compared to a work product describing it
 - t) measuring how well a system works in terms of its product characteristics
- a) v=q, w=s, x=r, y=t
 - b) v=s, w=t, x=r, y=q
 - c) v=r, w=s, x=t, y=q
 - d) v=s, w=q, x=r, y=t

4. Given the following decision table, what is the expected result for the test case listed below?

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
< 10 kg	True	True	False	False
< £10	True	False	True	False
Actions				
Must pay in cash only	True	False	True	False
Free delivery	False	False	False	True

What is the expected result for the following test case?

TC1: Purchase of a toaster weighing 3 kg, for £15.95

- a) Don't need to pay in cash, and no free delivery
- b) Need to pay in cash, and no free delivery
- c) Don't need to pay in cash, and free delivery
- d) Need to pay in cash, and free delivery

5. When the software contains an incorrect statement this is called:

- a) an error
- b) a defect
- c) a failure
- d) a problem

6. Equivalence partitioning is:

- a) a black box testing technique used only by developers
- b) a black box testing technique that can only be used during system testing
- c) a black box testing technique appropriate to all levels of testing
- d) a white box testing technique appropriate for component testing

Note: Questions 7 to 9 are based on the following pseudo-code. Draw the control flow chart if this will be helpful.

```

Read (Result)
IF Result < 60 THEN
    Print "Sorry but you have failed"
ELSE
    IF Result < 80 THEN
        Print "Well done – you have passed"
    ELSE
        Print "Excellent – you have been awarded
            a distinction"
    ENDIF
ENDIF
  
```

7. What is the minimum number of test cases needed to achieve 100% statement coverage?

- a) 1
- b) 2
- c) 3
- d) 4

8. What is the minimum number of test cases needed to achieve 100% decision coverage?

- a) 1
- b) 2
- c) 3
- d) 4

9. Which set of test cases below achieves 100% statement coverage?

- a) Result = 60, 80 and 100
- b) Result = 40, 70 and 90
- c) Result = 60 and 80
- d) Result = 50 and 70

10. White box tests are:

- a) normally derived by end users
- b) based on structural aspects of the system
- c) based on functional aspects of the system
- d) derived from a specification

11. Which of the following is NOT a dynamic testing technique?

- a) boundary value analysis
- b) technical reviews
- c) decision testing
- d) decision tables

12. Why are reviews good to do?

- a) they reduce costs by reducing the number of defects before test execution
- b) it means that dynamic testing will not be needed
- c) it allows people to meet members of the opposite sex in a neutral environment
- d) it means that test documentation is not needed

13. What is an invalid boundary?

- a) an input value that must not be entered by the user
- b) a maximum or minimum value in an invalid equivalence partition
- c) any value identified by boundary value analysis
- d) a value just less than a maximum valid partition value

14. What is an equivalence partition (sometimes known as an equivalence class)?

- a) a set of test cases for testing classes of objects
- b) an input or output range of values such that only one value in the range becomes a test case
- c) an input or output range of values such that each value in the range becomes a test case
- d) an input or output range of values such that every tenth value in the range becomes a test case

Note Questions 15 to 17 refer to exam scoring. A student needs to score at least 50 points to Pass. If they score at least 100 points they will achieve a Merit and if they score at least 150 points they will achieve a Distinction.

15. Which two values are in the same partition?

- a) 45 and 55
- b) 55 and 120
- c) 50 and 60
- d) 45 and 170

16. Which two boundary values are in the same partition?

- a) 49 and 50
- b) 50 and 150
- c) 50 and 99
- d) 50 and 100

17. Which of the following would be the most likely set of values identified by the Boundary Value Analysis technique?

- a) 49,50,99,100,149,150
- b) 0,49,150
- c) 49,50,149,150
- d) 50,100,150

18. Which type of tool is most often used for regression testing?

- a) Performance testing tool
- b) Static analysis tool
- c) Test execution tool
- d) Test harness

19. Which statement about integration testing is true?

- a) system integration testing tests the interactions between components
- b) there may be more than one level of integration testing
- c) integration testing is only carried out after component testing and before system testing
- d) integration testing must be carried out by an independent integration test team

20. What is the difference between static and dynamic testing?

- a) static testing does not execute the code, dynamic testing does execute the code
- b) static testing can tell what percentage of the code has been tested by dynamic testing
- c) static testing executes the code, dynamic testing does not execute the code
- d) static testing finds failures, dynamic testing finds defects

Practice Exam 4 Answers

1. What do walkthroughs, technical reviews and inspections have in common?

- I. They have defect finding as an objective.
- II. Pre-meeting preparation is required.
- III. They can be performed as a “peer review”.
- IV. The meeting is led by a trained moderator.

- a) I, II and III
- b) III and IV
- c) I and III
- d) II, III and IV

2. User Acceptance Testing is most appropriately performed by:

- a) a test team of software developers
- b) business users
- c) the project manager
- d) the support group or support staff

3. Given the following test types, which of the following is an objective for that type?

- v) functional
- w) non-functional
- x) structural
- y) change-related
 - q) defect fixes and unexpected side-effects
 - r) assesses the thoroughness of the testing
 - s) how the system works compared to a work product describing it
 - t) measuring how well a system works in terms of its product characteristics
- a) v=q, w=s, x=r, y=t
- b) v=s, w=t, x=r, y=q
- c) v=r, w=s, x=t, y=q
- d) v=s, w=q, x=r, y=t

4. Given the following decision table, what is the expected result for the test case listed below?

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
< 10 kg	True	True	False	False
< £10	True	False	True	False
Actions				
Must pay in cash only	True	False	True	False
Free delivery	False	False	False	True

What is the expected result for the following test case?

TC1: Purchase of a toaster weighing 3 kg, for £15.95

- a) Don't need to pay in cash, and no free delivery
- b) Need to pay in cash, and no free delivery
- c) Don't need to pay in cash, and free delivery
- d) Need to pay in cash, and free delivery

5. When the software contains an incorrect statement this is called:
- a) an error
 - b) a defect
 - c) a failure
 - d) a problem
6. Equivalence partitioning is:
- a) a black box testing technique used only by developers
 - b) a black box testing technique that can only be used during system testing
 - c) a black box testing technique appropriate to all levels of testing
 - d) a white box testing technique appropriate for component testing

Note: Questions 7 to 9 are based on the following pseudo-code. Draw the control flow chart if this will be helpful.

```

Read (Result)
IF Result < 60 THEN
    Print "Sorry but you have failed"
ELSE
    IF Result < 80 THEN
        Print "Well done – you have passed"
    ELSE
        Print "Excellent – you have been awarded
            a distinction"
    ENDIF
ENDIF
  
```

7. What is the minimum number of test cases needed to achieve 100% statement coverage?
- a) 1
 - b) 2
 - c) 3
 - d) 4
8. What is the minimum number of test cases needed to achieve 100% decision coverage?
- a) 1
 - b) 2
 - c) 3
 - d) 4
9. Which set of test cases below achieves 100% statement coverage?
- a) Result = 60, 80 and 100
 - b) Result = 40, 70 and 90
 - c) Result = 60 and 80
 - d) Result = 50 and 70
10. White box tests are:
- a) normally derived by end users
 - b) based on structural aspects of the system
 - c) based on functional aspects of the system
 - d) derived from a specification

11. Which of the following is NOT a dynamic testing technique?

- a) boundary value analysis
- b) technical reviews
- c) decision testing
- d) decision tables

12. Why are reviews good to do?

- a) they reduce costs by reducing the number of defects before test execution
- b) it means that dynamic testing will not be needed
- c) it allows people to meet members of the opposite sex in a neutral environment
- d) it means that test documentation is not needed

13. What is an invalid boundary?

- a) an input value that must not be entered by the user
- b) a maximum or minimum value in an invalid equivalence partition
- c) any value identified by boundary value analysis
- d) a value just less than a maximum valid partition value

14. What is an equivalence partition (sometimes known as an equivalence class)?

- a) a set of test cases for testing classes of objects
- b) an input or output range of values such that only one value in the range becomes a test case
- c) an input or output range of values such that each value in the range becomes a test case
- d) an input or output range of values such that every tenth value in the range becomes a test case

Note Questions 15 to 17 refer to exam scoring. A student needs to score at least 50 points to Pass. If they score at least 100 points they will achieve a Merit and if they score at least 150 points they will achieve a Distinction.

15. Which two values are in the same partition?

- a) 45 and 55
- b) 55 and 120
- c) 50 and 60
- d) 45 and 170

16. Which two boundary values are in the same partition?

- a) 49 and 50
- b) 50 and 150
- c) 50 and 99
- d) 50 and 100

17. Which of the following would be the most likely set of values identified by the Boundary Value Analysis technique?

- a) 49,50,99,100,149,150
- b) 0,49,150
- c) 49,50,149,150
- d) 50,100,150

18. Which type of tool is most often used for regression testing?

- a) Performance testing tool
- b) Static analysis tool
- c) Test execution tool
- d) Test harness

19. Which statement about integration testing is true?

- a) system integration testing tests the interactions between components
- b) there may be more than one level of integration testing
- c) integration testing is only carried out after component testing and before system testing
- d) integration testing must be carried out by an independent integration test team

20. What is the difference between static and dynamic testing?

- a) static testing does not run tests, dynamic testing does run tests through the code
- b) static testing can tell what percentage of the code has been tested by dynamic testing
- c) static testing runs tests, dynamic testing does not run tests through the code
- d) static testing finds failures, dynamic testing finds defects

Practice Exam 5

1. Which of the following is a project risk?

- a) the test environment is not a close enough match to the operational environment
- b) a large number of defects are found when the tests are first run in the test environment
- c) the test environment may not be available when needed
- d) the software may not be fast enough in one of the target environments

2. Given the following decision table

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions						
Online order	True	True	False	False	False	False
Coupon	True	False	True	True	False	False
> € 150	Don't care	Don't care	True	False	True	False
Actions						
Give discount	True	False	True	False	False	False
Free delivery	True	True	True	True	True	False

What is the expected result for each of the following test cases?

TC1: An online order for €50 with a coupon

TC2: A telephone order for €150 without a coupon

- a) TC1: discount and free delivery, TC2: no discount but free delivery
- b) TC1: discount but no free delivery, TC2: discount and free delivery
- c) TC1: discount and free delivery, TC2: discount and free delivery
- d) TC1: discount and free delivery, TC2: no discount and no free delivery

3. Which of the following are typical tasks of the Test Leader?

- 1) select the test approach and acquire resources
 - 2) take action based on monitoring of test progress
 - 3) create test specifications and review the test basis for testability
 - 4) set up the test environment
 - 5) write test summary reports
- a) 1, 2, 3 and 4
 - b) 1, 2 and 5
 - c) 2, 3 and 4
 - d) 1, 3 and 5

4. Unreachable code would best be found using:

- a) code inspections
- b) a coverage tool
- c) a test management tool
- d) a static analysis tool

5. Given the following state table

	A	B	C	D
SS	S1			
S1	ES	S2		
S2		ES	S3	
S3			ES	ES
ES				

Which of the following represents a VALID state transition?

- a) B from State S3
 - b) D from State S1
 - c) B from State S1
 - d) C from State S1
- 6. Which of the following would be included in a Test Summary Report?:**
- a) metrics about the effectiveness of the testing
 - b) the difference between the output of a test run and its expected output
 - c) the summary of all the values entered to a financial calculation
 - d) the management reporting structure for the next project
- 7. What information need not be included in a test incident report?**
- a) test environment details
 - b) how to fix the fault
 - c) severity, priority
 - d) the actual and expected outcomes
- 8. Match the following terms with the following definitions**
- q) Test execution tools
 - r) Dynamic analysis tools
 - s) Coverage measurement tools
 - t) Static analysis tools
 - u) Performance measurement tools
 - v) Monitors allocation, use and de-allocation of resources
 - w) Logging and recording of the number of transactions executed
 - x) Provides capture and replay facilities
 - y) Provides objective measures of the quality of the code
 - z) Programs are instrumented using these tools
- a) q=x; r=y; s=z; t=v; u=w
 - b) q=y; r=z; s=w; t=x; u=v
 - c) q=x; r=v; s=z; t=y; u=w
 - d) q=z; r=v; s=x; t=y; u=w
- 9. Which of the following is a potential benefit of using a tool to support testing?**
- a) no effort will be needed to maintain the test assets generated by the tool
 - b) repetitive work is reduced and there is greater consistency and repeatability of tests
 - c) the existing test process will not need to be changed
 - d) the tool itself will be well tested and free from defects

10. Which of the following would be done in a pilot project for a new tool?

1. decide on standard naming conventions and ways of using the tool
2. encourage all testers in the organization to use the tool
3. assess whether benefits will be achieved at reasonable cost
4. compare the tool to other similar tools from other tool vendors
5. investigate how existing test documentation and process should be changed

- a) 1, 3 and 5 are true, 2 and 4 are false
- b) 1, 2 and 4 are true, 3 and 5 are false
- c) 1, 3 and 4 are true, 2 and 5 are false
- d) 2, 4 and 5 are true, 1 and 3 are false

Practice Exam 5 Answers

1. Which of the following is a project risk?

- a) the test environment is not a close enough match to the operational environment
- b) a large number of defects are found when the tests are first run in the test environment
- c) the test environment may not be available when needed
- d) the software may not be fast enough in one of the target environments

2. Given the following decision table

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions						
Online order	True	True	False	False	False	False
Coupon	True	False	True	True	False	False
> € 150	Don't care	Don't care	True	False	True	False
Actions						
Give discount	True	False	True	False	False	False
Free delivery	True	True	True	True	True	False

What is the expected result for each of the following test cases?

TC1: An online order for €50 with a coupon

TC2: A telephone order for €150 without a coupon

- a) TC1: discount and free delivery, TC2: no discount but free delivery
- b) TC1: discount but no free delivery, TC2: discount and free delivery
- c) TC1: discount and free delivery, TC2: discount and free delivery
- d) TC1: discount and free delivery, TC2: no discount and no free delivery

3. Which of the following are typical tasks of the Test Leader?

- 1) select the test approach and acquire resources
 - 2) take action based on monitoring of test progress
 - 3) create test specifications and review the test basis for testability
 - 4) set up the test environment
 - 5) write test summary reports
- a) 1, 2, 3 and 4
 - b) 1, 2 and 5
 - c) 2, 3 and 4
 - d) 1, 3 and 5

4. Unreachable code would best be found using:

- a) code inspections
- b) a coverage tool
- c) a test management tool
- d) a static analysis tool

5. Given the following state table

	A	B	C	D
SS	S1			
S1	ES	S2		
S2		ES	S3	
S3			ES	S4
ES				

Which of the following represents a VALID state transition?

- a) B from State S3
- b) D from State S1
- c) B from State S1
- d) C from State S1

6. Which of the following would be included in a Test Summary Report?:

- a) metrics about the effectiveness of the testing
- b) the difference between the output of a test run and its expected output
- c) the summary of all the values entered to a financial calculation
- d) the management reporting structure for the next project

7. What information need not be included in a test incident report?

- a) test environment details
- b) how to fix the fault
- c) severity, priority
- d) the actual and expected outcomes

8. Match the following terms with the following definitions

- q) Test execution tools
 - r) Dynamic analysis tools
 - s) Coverage measurement tools
 - t) Static analysis tools
 - u) Performance measurement tools
 - v) Monitors allocation, use and de-allocation of resources
 - w) Logging and recording of the number of transactions executed
 - x) Provides capture and replay facilities
 - y) Provides objective measures of the quality of the code
 - z) Programs are instrumented using these tools
- a) q=x; r=y; s=z; t=v; u=w
 - b) q=y; r=z; s=w; t=x; u=v
 - c) q=x; r=v; s=z; t=y; u=w
 - d) q=z; r=v; s=x; t=y; u=w

9. Which of the following is a potential benefit of using a tool to support testing?

- a) no effort will be needed to maintain the test assets generated by the tool
- b) repetitive work is reduced and there is greater consistency and repeatability of tests
- c) the existing test process will not need to be changed
- d) the tool itself will be well tested and free from defects

10. Which of the following would be done in a pilot project for a new tool?

1. decide on standard naming conventions and ways of using the tool
2. encourage all testers in the organization to use the tool
3. assess whether benefits will be achieved at reasonable cost
4. compare the tool to other similar tools from other tool vendors
5. investigate how existing test documentation and process should be changed

- a) 1, 3 and 5 are true, 2 and 4 are false
- b) 1, 2 and 4 are true, 3 and 5 are false
- c) 1, 3 and 4 are true, 2 and 5 are false
- d) 2, 4 and 5 are true, 1 and 3 are false

Practice Exam 1_2 Questions

1 Which of the following examples BEST describes an effect of software defects?

- A The developer made a serious mistake due to defects in the code.
- B A defect caused a power surge resulting in a system failure.
- C Usability defects in the web application resulted in a loss of business.
- D The system was slow due to insufficient bandwidth in the network.

K2 (1 mark)

2 Which of the following statements about the causes and effects of a software defect is correct?

- A The use case contained an error because not all actors were specified.
- B The developer made a mistake when coding causing an error at run time.
- C A failure in the code logic caused a defect during execution.
- D Specification defects resulted from the analyst's misunderstanding.

K2 (1 mark)

3 Which of the following describes why it is necessary to test?

- A To reduce the risk of problems occurring in the production system.
- B To ensure that software is bug free before live deployment.
- C To help improve industry specific standards.
- D To show that the code meets the developer's expectations.

K2 (1 mark)

4 Following the execution of a suite of UAT test cases, no new defects were found in the enhanced ordering system. What does this tell us about the quality of the system under test?

- A The system quality must be excellent and the product is ready for shipping.
- B We cannot say that the product is ready to ship unless we know the test cases were of sufficient quality.
- C The system cannot be shipped yet as UAT has clearly failed to meet its test objectives.
- D As there has been no defect fixing, the quality of the system has not increased sufficiently.

K2 (1 mark)

- 5** An on-line shopping application includes a loyalty rewards scheme whereby the customer accumulates rewards points based on the total amount of each shopping order.

A problem was detected during testing whereby items that were removed from the shopping cart before purchase still accumulated points. Debugging identified the component which caused the problem and further analysis showed that the developer had misunderstood the requirement.

Based on this scenario, which of the following correctly best describes the terms used when describing the cause of software defects?

- A** The failure identified by debugging meant that points accumulation was no longer at fault.
- B** A bug in the component caused a defect during points accumulation.
- C** The developer made a mistake during coding which subsequently caused the application to fail.
- D** The requirement contained an error which caused a bug to be introduced into the accumulation module.

K2 (1 mark)

- 6** Which of the following is not a valid objective of testing?

- A** Preventing defects from being introduced into the code.
- B** Gaining confidence that the system has reached the desired level of quality.
- C** Providing adequate information for the stakeholders to make important decisions.
- D** Locating and fixing defects in the code.

K1 (1 mark)

- 7** Which of the following would represent valid objectives for user acceptance testing an on-line hotel booking system?

- i. To confirm that hotel bookings work as expected.
- ii. To identify defects in the web-site user interface.
- iii. To gain confidence that hotel booking requirements have been met.
- iv. To assess the reliability of the hotel booking application.
- v. To ensure no new defects have been introduced as a result of recent changes made to hotel bookings.

- A** i and v
- B** ii and iii
- C** i and iii
- D** iv and v

K2 (1 mark)

8 Consider the following testing and debugging activities:

- i. Locating the cause of the failure.
- ii. Checking the fix has resolved the failure.
- iii. Fixing the defect.
- iv. Identifying a failure.

Which of the following correctly allocates the roles of tester and developer to the activities and also places them in the correct sequence (starting at 1 and ending with 4)?

- A** 1. Tester - iv; 2. Developer - i; 3. Developer - iii; 4. Tester - ii
- B** 1. Tester - iv; 2. Developer - i; 3. Developer - iii; 4. Developer - ii
- C** 1. Developer - i; 2. Developer - iii; 3. Developer - ii; 4. Tester - iv
- D** 1. Tester - i; 2. Developer - ii; 3. Developer - iii; 4. Tester - iv

K2 (1 mark)

9 The full suite of tests for a release have been run and, following defect fixing and successful re-testing, all have now been marked as 'passed'.

Which of the following statements is generally true in this situation?

- A** There are no more defects in the system, further testing is unnecessary.
- B** There may be further defects in the system, further testing will find them.
- C** There may be further defects in the system, further testing may not be necessary.
- D** If the tests covered all of the requirements then the system will meet the users expectations.

K2 (1 mark)

10 During which stage of the fundamental test process does the tester evaluate whether the requirements are testable?

- A** Test Planning
- B** Test Analysis and Design
- C** Test Implementation and Execution
- D** Evaluating Exit Criteria and Reporting

K1 (1 mark)

11 Which of the following would typically be the most effective way of finding defects and failures?

- A** Teaching the developers good test design to eliminate the need for costly external testers.
- B** Ensure the test team is located separately to avoid developer influence.
- C** Having an independent test team responsible for all levels of testing.
- D** Having an independent test team responsible for some levels of testing.

K1 (1 mark)

12 Which of the following best describes why it is better to have a certain degree of independence in testing?

- A** Independent testers understand the requirements better than developers.
- B** Independent testers design tests faster and cheaper than developers.
- C** Developers want their code to work, so may miss bugs.
- D** Developers don't possess the same attention to detail as testers.

K2 (1 mark)

Practice Exam 1_2 Answers

CTFL-1.1.1 K2 Describe, with examples, the way in which a defect can cause harm to a person, to the environment or to a company.

1 The answer is B.

- A NO - mistakes result in defects, not the other way around.
- B NO - a power surge is an (external) environmental condition that can cause failures. It is very unlikely to result from a software defect.
- C YES - an example of loss of business reputation/loss of money.
- D NO - more likely a hardware issue rather than a software defect.

CTFL-1.1.2 K2 Distinguish between the root cause of a defect and its effects

2 The answer is D.

The sequence is: a human make an error/mistake that may result in a defect (in documentation or code) that when executed may cause the software to fail.

- A NO - it would contain a defect, not an error.
- B NO - it would cause a failure at run time.
- C NO - wrong way around - defects cause failures.
- D YES - mistakes/errors leading to defects in the documentation.

CTFL-1.1.3 K2 Give reasons why testing is necessary by giving examples

3 The answer is A.

- A YES. Word for word per the syllabus.
- B NO. Contradicts the key principle of testing - 'Testing shows the presence of defects' No such things as a bug free system.
- C NO. Per syllabus, testing may be required to meet industry specific standards, not improve them.
- D NO. The wrong mind set for testing (allied with section 1.5 in the syllabus).

CTFL-1.1.4 K2 Describe why testing is part of quality assurance and give examples of how testing contributes to higher quality.

4 The answer is B.

- A NO. Per B, we need to know how good the tests are first before we can make such a statement.
- B YES. Per syllabus 'A properly designed test case that passes reduces the overall level of risk in the system'.
- C NO. The objective of UAT is generally not to find defects but to gain confidence. It depends on the quality of the tests that were run. If the tests were fine, then the product may be OK to ship.
- D NO. This misinterprets the meaning of the statement in the syllabus which says 'when testing does find defects, the quality of the software system increases when those defects are fixed'.

CTFL-1.1.5 K2 Explain and compare the terms error, defect, fault, failure, and the corresponding terms mistake and bug, using examples.

5 The answer is C.

- A NO. Debugging identifies defects/faults/bugs (the causes of failures) rather than failures themselves.
- B NO. Bug and defect are the same thing. The latter should say failure.
- C YES. Although there is an interim 'stage', i.e. the defect caused by the mistake, it is correct to say that the mistake was the original cause of the failure.
- D NO. Although this is the correct sequence of events, i.e. error leading to a bug, this does not support the scenario which says the developer made a mistake interpreting the requirement, so the requirement itself was not at fault.

CTFL-1.2.1 K1 Recall the common objectives of testing.

6 The answer is D.

- A YES. 'Preventing Defects' (e.g. requirements reviews).
- B YES. 'Gaining confidence about the level of quality'.
- C YES. 'Providing information for decision making'.
- D NO. This is debugging, not testing.

CTFL-1.2.2 K2 Provide examples for the objectives of testing in different phases of the software lifecycle.

7 The answer is C.

i and iii are per the syllabus 'In acceptance testing, the main objective may be to confirm that the system works as expected, to gain confidence that it has met the requirements'.

ii seems attractive as it mentions the user interface, which is often a focus of UAT, but UAT is not primarily aimed at finding defects. This should be objective for development testing.

iv is a typical objective for operational (acceptance) testing, per syllabus 'During operational testing, the main objective may be to assess system characteristics such as reliability or availability'.

v is an objective of maintenance testing, per syllabus 'Maintenance testing often includes testing that no new defects have been introduced during development of the changes'.

CTFL-1.2.3 K2 Differentiate testing from debugging.

8 The answer is A.

1. The Tester identifies a failure (iv).
2. The Developer locates the cause of the failure, i.e. the defect (i).
3. The Developer fixes the defect (iii).
4. The Tester checks that the fix has resolved the failure (ii).

CTFL-1.3.1 K2 Explain the seven principles in testing.

9 The answer is C.

This question addresses two of the key principles of testing:
 Principle 1 – Testing shows presence of defects.

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

Principle 7 – Absence-of-errors fallacy.

Finding and fixing defects does not help if the system built is unusable and does not fulfil the users' needs and expectations.

- A NO. Contradicts principle 1. Testing cannot prove that there are no defects.
- B NO. Contradicts principle 1. Testing can, not will, show that defects are present. Even with the best designed tests, it can never guarantee to find all the defects.
- C YES. Further testing is a risk based decision.
- D NO. Contradicts principle 7. If the end-users were not involved in agreeing the requirements and testing the system (e.g. during UAT) then the system is unlikely to meet the users expectations.

CTFL-1.4.1 K1 Recall the five fundamental test activities and respective tasks from planning to closure.

10 The answer is B.

Per syllabus, "The test analysis and design activity has the following major tasks:

- Reviewing the test basis (such as requirements, software integrity level (risk level), risk analysis reports, architecture, design, interface specifications).
- Evaluating testability of the test basis and test objects."

CTFL-1.5.1 K1 Recall the psychological factors that influence the success of testing.

11 The answer is D.

- A NO. Although teaching developers good testing practice is a good idea, it is more effective to have some levels of testing designed by external testers (see C).
- B NO. This mindset is likely to result in communication issues. Testers and developers need to communicate effectively.
- C NO. Developers should have testing responsibilities (e.g. component testing and component integration testing) otherwise they are likely to lose a sense of quality. It is also likely to be inefficient if independent testers conducted component testing.
- D YES. Per syllabus 'A certain degree of independence (avoiding the author bias) often makes the testing more effective ... Independence is not a replacement for familiarity.' makes this the most effective of the 4 options.

12 The answer is C.

- A NO. Both roles should need to possess an equal understanding of the requirements.
- B NO. Not true, particularly for early test levels such as component testing.
- C YES. The natural mindset of a developer is of wanting the code to work. This will mean that they may not be as effective in designing tests that try to identify failures.
- D NO. Although this is one of a number of an important attributes for testers (per syllabus) it is not unique to them. Attention to detail is certainly important in software design too.

Practice Exam 2_2 Questions

- 1 Which of the following is usually true for a project using a V-model software development lifecycle?
- A The specification documents produced for any development level can be used by any test level.
 - B Acceptance test cases should be the last to be designed in the project.
 - C System design should be validated against the system requirements.
 - D System test design and implementation can be completed once the requirements have been agreed.

K2 (1 mark)

- 2 If a master test plan required there to be a system integration test level, where should it be placed in a V-model diagram?
- A Between component testing and component integration testing
 - B Between component testing and system testing
 - C Between component integration testing and system testing
 - D Between system testing and acceptance testing

K1 (1 mark)

- 3 Which of the following represents good testing practice for test analysis and design in a given test level, irrespective of the software lifecycle model used? f identicno so narednoto prasanje
- A It should begin before the corresponding development level completes
 - B It should begin as soon as the corresponding development level completes
 - C It should start at the same time as the corresponding development level starts
 - D It should have a specific objective to only use the outputs from the corresponding development level

K1 (1 mark)

- 4 Which of the following represents good testing practice for test analysis and design in a given test level, irrespective of the software lifecycle model used? f identicno so predhodnoto prasanje
- A It should begin before the corresponding development level completes.
 - B It should begin as soon as the corresponding development level completes.
 - C It should start at the same time as the corresponding development level starts.
 - D It should have a specific objective to only use the outputs from the corresponding development level.

K1 (1 mark)

- 5 Which of the following statements is true for component integration testing?
- A Increasing the number of components in the integration environment leads to more effective defect identification.
 - B The test basis may include software architecture and use case models.
 - C It generally tests the interactions between different systems or between hardware and software.
 - D Non-functional testing for a project starts as soon as component integration testing completes.

K2 (1 mark)

6 An emergency maintenance release for an on-line shopping system consisted of code fixes for two separate live problems. The following existing system test cases were executed to ensure that the defects were successfully removed:

- Test case 1 - To demonstrate that customer's payment details could not be accessed by malicious outsiders.
- Test case 2 -To demonstrate that the card expiry date is a mandatory field.

Which of the following are test types that would have applied for these test cases?

- i. Functional testing
- ii. Non-functional testing
- iii. Structural testing
- iv. Re-testing
- v. System testing

- A** ii, iv
- B** i, iv
- C** i, iii, v
- D** ii, iv, v

K2 (1 mark)

7 For which of the following test levels can functional and structural testing both be applied?

- A** Component and component integration testing only
- B** Component and system testing only
- C** All test levels before acceptance testing
- D** All test levels

K1 (1 mark)

8 Which of the following represent non-functional tests for an airline booking system?

- i. All available flights are displayed in less than 5 seconds.
- ii. Either the 'Return' or 'One Way' radio button must be selected.
- iii. The flight booking screen is easy to use for customers.
- iv. Flights can be paid for securely.

- A** i and iv
- B** all of the above
- C** i and iii
- D** ii and iv

K2 (1 mark)

9 Which of the following is a measure of structural test coverage?

- A** To ensure that all external system interfaces have been tested.
- B** To ensure that all defect fixes have been re-tested.
- C** To ensure that all performance targets have been tested.
- D** To ensure that all decision branches have been tested.

K2 (1 mark)

10 Which of the following statements is NOT true for regression testing?

- A** Regression tests are good candidates for test automation.
- B** Regression testing is an important consideration during maintenance testing.
- C** Regression testing applies for already tested software.
- D** Regression testing confirms that the original defect has been successfully removed.

K2 (1 mark)

11 Which of the following statements best describes maintenance testing?

- A** It is a non-functional type of testing to establish that code can be easily changed in the future.
- B** It is triggered by changes to the software or the environment before they are first delivered into the live service.
- C** It is triggered by changes to an operational system and uses impact analysis to determine the scope of regression testing.
- D** It applies extensive regression testing to the parts of the system that have been changed.

K2 (1 mark)

12 Which of the following would trigger the need for maintenance testing?

- i. Planned enhancements to the layout of the existing operational screens
- ii. Migration of data from another application to the existing live database
- iii. An emergency fix to resolve a live problem
- iv. An upgrade to the COTS product being used in the production system

- A** i, ii and iii
- B** All of the above
- C** i and iv
- D** ii, iii and iv

K1 (1 mark)

13 Which of the following is true?

- A** Impact analysis is used to help decide how much regression testing to do for an existing system.
- B** Impact analysis assesses the effect that a defect found in regression testing has on a live system.
- C** Impact analysis is used to assess how many regression tests should be automated.
- D** Impact analysis is used to re-evaluate code coverage when a new test is added to a suite.

K2 (1 mark)

Practice Exam 2_2 Answers

CTFL-2.1.1 K2 Explain the relationship between development, test activities and work products in the development life cycle, by giving examples using project and product types.

1 The answer is C.

- A NO. For example, program specifications are unlikely to be used for Acceptance test design.
- B NO. They should be the first to be designed (or at least started), as soon as the business requirements are stable/agreed.
- C YES. Validation involves ensuring that the output from a development level is consistent with the output from the previous level.
- D NO. System test, like most test levels, requires the output from more than one development level. System test implementation cannot complete until the system design has been agreed. System design may necessitate further test cases (e.g. testing of a batch scheduling system) and provides the detail needed to complete the test procedures.

CTFL-2.1.2 K1 Recognize the fact that software development models must be adapted to the context of project and product characteristics

2 The answer is D.

Per syllabus - In practice, a V-model may have more, fewer or different levels of development and testing, depending on the project and the software product. For example, there may be component integration testing after component testing, and system integration testing after system testing.

CTFL-2.1.3 K1 Recall characteristics of good testing that are applicable to any life cycle model.

3 The answer is A. f identicno so narednoto prasanje

- A YES. Per syllabus 'The analysis and design of tests for a given test level should begin during the corresponding development activity'.
- B NO. Per A, before not after.
- C NO. Almost certainly too early as there will be nothing of use to analyse.
- D NO. A distortion of the good testing practice 'Each test level has test objectives specific to that level'. Test level design often needs the outputs of several development levels (e.g. UAT test cases may require systems requirements as well as business requirements).

CTFL-2.1.3 K1 Recall characteristics of good testing that are applicable to any life cycle model.

4 The answer is A. f identicno so predhodnoto prasanje

- A YES. Per syllabus 'The analysis and design of tests for a given test level should begin during the corresponding development activity'.
- B NO. Per A, before not after.
- C NO. Almost certainly too early as there will be nothing of use to analyse.
- D NO. A distortion of the good testing practice 'Each test level has test objectives specific to that level'. Test level design often needs the outputs of

several development levels (e.g. UAT test cases would require systems requirements as well as business requirements).

CTFL-2.2.1 K2	Compare the different levels of testing: major objectives, typical objects of testing, typical targets of testing (e.g., functional or structural) and related work products, people who test, types of defects and failures to be identified.
---------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5 The answer is B.

- A NO. The opposite. Per syllabus 'In order to ease fault isolation and detect defects early, integration should normally be incremental rather than “big bang”.' Therefore the more components being tested, the harder it is to isolate a bug.
- B YES. Note that Use Cases can also be used to shape top-down integration approaches (e.g. including all components that make up a use case scenario/path in the integration environment). Per syllabus in section 4.3.5 Use Cases 'also help uncover integration defects caused by the interaction and interference of different components, which individual component testing would not see'.
- C NO. This refers to system integration testing.
- D NO. Maybe true in some projects but this is bad practice. Per syllabus 'Testing of specific non-functional characteristics (e.g., performance) may be included in integration testing as well as functional testing.' Indeed non-functional testing can start as early as component testing.

CTFL-2.3.1 K2	Compare four software test types (functional, non-functional, structural and change related) by example.
---------------	----------------------------------------------------------------------------------------------------------

6 The answer is B.

Test case 1 - This is security testing, a functional test type (i).

Test case 2 - This is a functional test, i.e. what the system is supposed to do (i).

As these were live defect fixes then re-testing (iv) would have applied (and likely to be regression testing but this is not on the list).

Non-functional testing (ii) does not apply (although some may think security testing is non-functional) and as this is system testing then structural testing (iii) is unlikely to be relevant (and neither test case suggests menu structures or similar are relevant tests) . System testing (v) is a test level, not a test type.

CTFL-2.3.2 K1	Recognize that functional and structural tests occur at any test level.
---------------	-------------------------------------------------------------------------

7 The answer is D.

Simple rule of thumb is that any test type can be applied at any test level, so D is correct.

CTFL-2.3.3 K2 Identify and describe non-functional test types based on non-functional requirements

8 The answer is C.

- i. Performance testing - Non-functional
- ii. A functional test (mandatory field validation)
- iii. Usability testing - Non-functional
- iv. Security testing - functional

CTFL-2.3.4 K2 Identify and describe test types based on the analysis of a software system's structure or architecture.

9 The answer is D.

- A Functional (interoperability).
- B Not a test type - just a generic test objective or exit criterion.
- C Non-functional.
- D Structural - Decision testing is a structural test technique.

CTFL-2.3.5 K2 Describe the purpose of confirmation testing and regression testing.

10 The answer is D.

- A TRUE - per syllabus 'Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation'.
- B TRUE - per syllabus 'In addition to testing what has been changed, maintenance testing includes extensive regression testing to parts of the system that have not been changed'.
- C TRUE - per syllabus 'Regression testing is the repeated testing of an already tested program'.
- D NOT TRUE - This is Re-testing.

CTFL-2.4.1 K2 Compare maintenance testing (testing an existing system) to testing a new application with respect to test types, triggers for testing and amount of testing.

11 The answer is C.

- NO - This is maintainability testing.
- NO - After the system goes live, not before.
- YES - per syllabus 'Maintenance testing is done on an existing operational system' and 'impact analysis is used to help decide how much regression testing to do.'
- NO - applies to the parts that have NOT been changed.

CTFL-2.4.2 K1 Recognize indicators for maintenance testing (modification, migration and retirement).

12 The answer is B.

Per syllabus:

'Modifications include planned enhancement changes (e.g., release-based), corrective and emergency changes, planned upgrade of Commercial-Off-The-Shelf software' – that is, i, iii and iv respectively.

'Migration testing (conversion testing) is also needed when data from another application will be migrated into the system being maintained.' that is, ii.

So all apply (hence B).

CTFL-2.4.3. K2 Describe the role of regression testing and impact analysis in maintenance.

13 The answer is A.

Only A is correct. Per syllabus 'Determining how the existing system may be affected by changes is called impact analysis, and is used to help decide how much regression testing to do'.

Practice Exam 3_2 Questions

1 Which of the following products would normally be examined using static analysis?

- A A requirement specification.
- B A test plan.
- C A software model.
- D Hardware components.

K1 (1 mark)

2 Which of the following best describes a benefit of applying static techniques?

- A They enable defects to be found during the early stages of test execution.
- B They can reduce testing cost and time.
- C They can minimise requirement change.
- D They can find failures in code logic.

K2 (1 mark)

3 Which of the following types of coding defects are effectively found by a formal review, rather than by other static or dynamic techniques?

- i. The requirement to limit the booking to 5 passengers was missing from the code logic.
- ii. There was a lack of comments in the code making it difficult to understand.
- iii. The flight reservation component ran too slowly.
- iv. There were a number of control and data flow anomalies.
- v. The tabbing sequence for reservation fields was illogical.

- A i, iv
- B ii, iii
- C i, ii
- D i, ii, iv, v

K2 (1 mark)

4 During which stage of an inspection process would the moderator explain the review procedures to the participants?

- A Kick Off
- B Planning
- C Individual Preparation
- D Explanation

K1 (1 mark)

5 Consider the following formal review main objectives:

- i. Solving technical problems
- ii. Evaluating alternatives
- iii. Finding defects
- iv. Gaining understanding

And the following formal review types:

- Walkthrough
- Technical review

- Inspection

Which option below correctly pairs the objectives and review types?

- A** Walkthrough - iv; Technical Review - ii; Inspection - i, iii.
- B** Walkthrough - ii; Technical Review - iii, iv; Inspection - i, iii.
- C** Walkthrough - iii, iv; Technical Review - i, ii, iii; Inspection - iii only.
- D** All objectives apply to each review type.

K2 (1 mark)

6 Which of the following options does NOT contain factors that help a review to be successful?

- A** Each review has clear predefined objectives for which the right people are selected and involved.
- B** There is management presence in each review meeting to ensure there is adequate time for the meeting to succeed.
- C** Defects found are welcomed and expressed objectively and there is an emphasis on learning and process improvement.
- D** Review techniques are applied that are suitable to achieve the objectives and training is provided for them.

K2 (1 mark)

7 Which of the following type of coding defect would be typically found by a static analysis tool?

- A** Lack of comments explaining code logic.
- B** Memory leakage.
- C** An infinite loop in a 'While' statement.
- D** Inconsistent interfaces between systems.

K1 (1 mark)

8 Which of the following statements best describes how static analysis adds benefit to the development process?

- A** Static analysis enables defects to be found well before the test environments are ready.
- B** Static analysis tools enable the project to develop efficient coding standards.
- C** If performed effectively, dynamic testing would not be required resulting in significant cost savings.
- D** Static analysis provides an early warning about suspicious members of the development team.

K2 (1 mark)

- 9** Which of the following would be typically found by using a static analysis tool?
- i. That the code defines a variable but does not make use of it.
 - ii. That the wrong value was passed into a variable from the calling component.
 - iii. That the program specification did not define the component parameters.
 - iv. That a variable has been incorrectly declared as an integer in the code.
- A** i, ii, iv
B ii, iii
C All of the above
D i, iv

K1 (1 mark)

Practice Exam 3_2 Answers

CTFL-3.1.1 K1 Recognize software work products that can be examined by the different static techniques.

1 The answer is C.

- A NO. By a review, not static analysis.
- B NO. By a review, not static analysis.
- C YES - Per syllabus 'The objective of static analysis is to find defects in software source code and software models'.
- D NO - a distracter; static suggests electrical currents.

CTFL-3.1.2 K2 Describe the importance and value of considering static techniques for the assessment of software work products.

2 The answer is B.

- A NO - although static testing is early testing, test execution is dynamic testing.
- B YES - per syllabus 'Benefits of reviews include early defect detection and correction, development productivity improvements, reduced development timescales, reduced testing cost and time'.
- C NO - They can improve the quality of requirements through reviews, but can't prevent them changing.
- D NO - dynamic testing finds failures, whereas static techniques find defects rather than the failures themselves.

CTFL-3.1.3 K2 Explain the difference between static and dynamic techniques, considering objectives, types of defects to be identified, and the role of these techniques within the software life cycle.

3 The answer is C.

- i. YES - 'Reviews can find omissions, for example, in requirements, which are unlikely to be found in dynamic testing'.
- ii. YES - Typical defects that are easier to find in reviews than in dynamic testing include insufficient maintainability.
- iii. NO - A dynamic performance failure.
- iv. NO - Much more likely to be found by a static analysis tool - 'Static analysis tools analyse program code (e.g., control flow and data flow)'.
- v. NO- a dynamic usability failure.

CTFL-3.2.1 K1 Recall the activities, roles and responsibilities of a typical formal review.

4 The answer is A.

Per syllabus, during the Kick-off stage, the leader/moderator would be responsible for:

- distributing documents,
- explaining the objectives, process and documents to the participants.

Option D is a distracter, there is no such stage in the inspection process.

CTFL-3.2.2 K2 Explain the differences between different types of reviews: informal review, technical review, walkthrough and inspection.

5 The answer is C.

The correct pairings are:

- i. Solving technical problems - Technical review
- ii. Evaluating alternatives - Technical review
- iii. Finding defects - All review types
- iv. Gaining understanding - Walkthrough

CTFL-3.2.3 K2 Explain the factors for successful performance of reviews.

6 The answer is B.

Options A, C and D each amalgamate 2 of the success factors listed in the syllabus (3.2.4).

Option B is incorrect. The syllabus says that 'Management supports a good review process (e.g., by incorporating adequate time for review activities in project schedules)', however management participation is not mandatory for any review type, so presence in the meeting is not required.

CTFL-3.3.1 K1 Recall typical defects and errors identified by static analysis and compare them to reviews and dynamic testing.

7 The answer is C.

- A NO. Would be found at a review meeting.
- B NO. Would be found during dynamic testing.
- C YES. Per syllabus 'Missing and erroneous logic (potentially infinite loops)'.
- D NO. Should be 'Inconsistent interfaces between modules and components'.

CTFL-3.3.2 K2 Describe, using examples, the typical benefits of static analysis.

8 The answer is A.

- A TRUE - Per syllabus 'Early detection of defects prior to test execution'.
- B FALSE - However they can verify code against standards.
- C FALSE - Dynamic testing finds different types of defects and is still needed.
- D FALSE - It should say 'Early warning about suspicious aspects of the code'.

CTFL-3.3.3 K1 List typical code and design defects that may be identified by static analysis tools.

9 The answer is D.

- i. YES. Per syllabus 'Variables that are not used or are improperly declared'.
- ii. NO. Would only occur at run time (dynamic testing).
- iii. NO. A specification defect, so found by a review.
- iv. YES. Per syllabus 'Syntax violations of code and software models'.

Practice Exam 4_2 Questions

1 Which of the following documents would be produced during test analysis to identify the test items, i.e. 'what' we want to test?

- A** Test plan
- B** Test design specification
- C** Test case specification
- D** Test procedure specification

K2 (1 mark)

2 Which of the following best describes the purpose of a test case?

- A** To define the objectives for a particular test level.
- B** To define the step by step instructions for the execution of a test.
- C** To define the input values and expected results for a particular objective.
- D** To define the order in which the tests should be executed.

K2 (1 mark)

3 Consider the following objectives:

- i. To enable effective impact analysis when requirements change
- ii. To achieve optimum use of test management and requirements management tools
- iii. To make it easier to automate the tests
- iv. To reduce the possibility of accepting incorrect results at run time
- v. To produce effective metrics for the stakeholders

Which two objectives explain why, when developing test cases, it is important to:

- maintain clear traceability to the requirements
- define expected results?

- A** Traceability - ii; Expected results - iii
- B** Traceability - i; Expected results - iii
- C** Traceability - v; Expected results - iv
- D** Traceability - i; Expected results - iv

K2 (1 mark)

- 4 An on-line banking system allows the customer to make payments to new or existing recipients. A test case has been designed to validate payments to existing recipients. The description of the test case reads:

'The customer selects an existing recipient from the drop-down list, then enters the payment amount into an 8 character field, which must contain a decimal point and 2 decimal places, and finally selects the Make Payment button. An appropriate error message should be immediately displayed if the format of the payment field is incorrect.'

Which of the following test procedures meets the objectives of this test case and would be most effective for a member of the test execution team who has no prior experience with the function under test?

- A Select first recipient from drop-down list, enter 9999.00 in Payment field, select Make Payment, confirm payment successful.
- B Select one recipient from drop-down list, enter 123.5 in Payment field, check correct error message, enter 123.50 in Payment field, select Make Payment, confirm payment successful.
- C Select any recipient from drop-down list, make an invalid entry in Payment field, check correct error message is displayed per requirement, make a valid entry in Payment field, select Make Payment, confirm payment successful on screen.
- D Enter 10000 in Payment field, check correct error message, select a recipient from the drop-down list, enter 10000.00 in Payment field, select Make Payment, confirm payment successful.

K3 (1 mark)

- 5 Which of the following techniques can be used to design tests by analysing the structure of a component or a system?

- A Decision table testing
- B Use case testing
- C Decision testing
- D State transition testing

K2 (1 mark)

- 6 You responsible for the functional testing of a system that has no useful test basis documentation available. Furthermore there is little time available for test design. Which of the following test design techniques or approaches would be most appropriate in the circumstances?

- A Use case testing
- B Exploratory testing
- C Decision testing
- D Risk-based testing

K2 (1 mark)

- 7** A booking system for a popular outdoor tourist attraction allows the customer to buy 'day tickets' that are priced according to the time of year. When the customer clicks the Date button, a standard 12 month/365 day calendar is presented for the current year (i.e. starting 1st January and ending 31st December) from which the day of visit is chosen.

A Spring tariff applies for day visits between the first day of March and the last day of May inclusive; a Summer tariff applies for day tickets between the first day of June and the last day of August inclusive; an Autumn tariff applies for day tickets between the first day of September and the last day of November inclusive. The attraction is closed for the remaining time of the year, the winter season, and is therefore not available for bookings.

Which of the following represents a set of test cases that shows that the equivalence partition test design technique has been used correctly for the above scenario? (Note: assume that the current year is not a leap year)

- A** 12th January; 1st April; 1st September; 1st December
- B** 1st February; 20th March; 3rd June; 19th September; 25th December
- C** 1st March; 31st May; 1st June; 31st August; 1st September; 30th November; 1st December; 28th February;
- D** 10th May; 10th July; 10th October

K3 (1 mark)

- 8** A booking system for a popular outdoor tourist attraction allows the customer to buy 'day tickets' that are priced according to the time of year. When the customer clicks the Date button, a standard 12 month/365 day calendar is presented for the current year (i.e. starting 1st January and ending 31st December) from which the day of visit is chosen.

A Spring tariff applies for day visits between the first day of March and the last day of May inclusive; a Summer tariff applies for day tickets between the first day of June and the last day of August inclusive; an Autumn tariff applies for day tickets between the first day of September and the last day of November inclusive. The attraction is closed for the remaining time of the year, the winter season, and is therefore not available for bookings.

In applying the boundary value analysis technique to the above scenario, which of the following represents a set of valid boundary values? (Note: assume that the current year is not a leap year.)

- A** 1st January; 28th February; 1st March; 31st May; 1st June; 31st August; 1st September; 30th November; 1st December; 31st December.
- B** 1st March; 30th May; 2nd June; 30th August; 2nd September; 30th November.
- C** 1st March; 31st May; 1st June; 31st August; 1st September; 30th November.
- D** January; February; March; May; June; August; September; November; December.

K3 (1 mark)

- 9** Consider the decision table below that reflects an airline's business rules for applying charges to customers checking in hold baggage onto an aeroplane. (Note that this is an incomplete decision table, i.e. there are logically more rules that could be shown.)

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Conditions					
Gold Card holder	Y	N	N	N	N

Business Class	Don't care	Y	Y	N	N
Frequent Flyer	Don't care	Y	N	Y	N
Baggage > 20KG	Don't care	Y	Y	Y	Y
Actions					
No extra charge	Y	Y	N	N	N
£5 fixed charge	N	N	Y	N	N
£10 fixed charge	N	N	N	Y	N
£15 per extra KG	N	N	N	N	Y

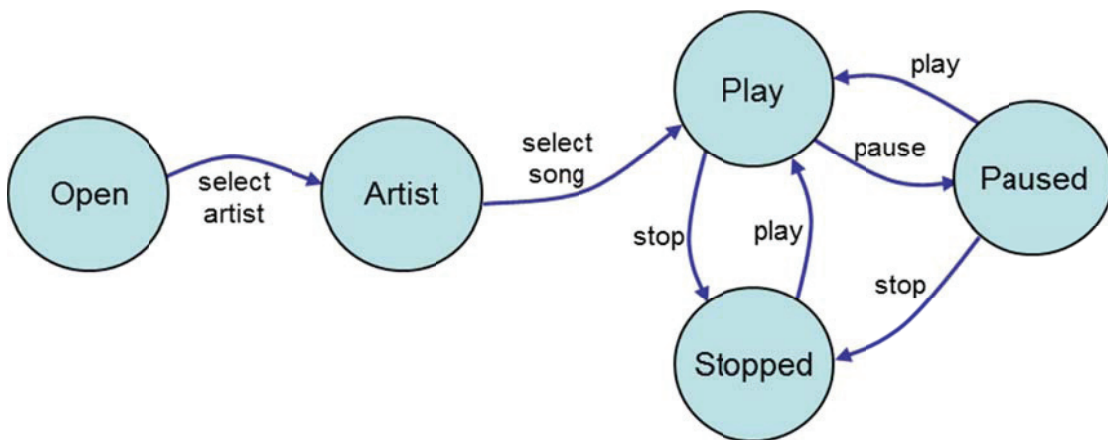
What are the expected results (actions) for each of the following two test cases (TC1 and TC2)?

- TC 1 - Roger is a frequent flyer, travelling Economy Class, doesn't have a gold card and is checking in a 23KG bag.
- TC 2 - Carol is flying Business Class, is not a frequent flyer, does have a gold card and is checking in a 30 KG bag.

- A** TC1 - no extra charge; TC2 - no extra charge.
B TC1 - £10 fixed charge; TC2 - £5 per extra KG.
C TC1 - £15 per extra KG; TC2 - £10 fixed charge.
D TC1 - £10 fixed charge; TC2 - no extra charge.

K3 (1 mark)

- 10** The State Transition diagram below reflects a music application on a Smartphone.



If the application was currently in the Open State and the following sequence of events then occurred:

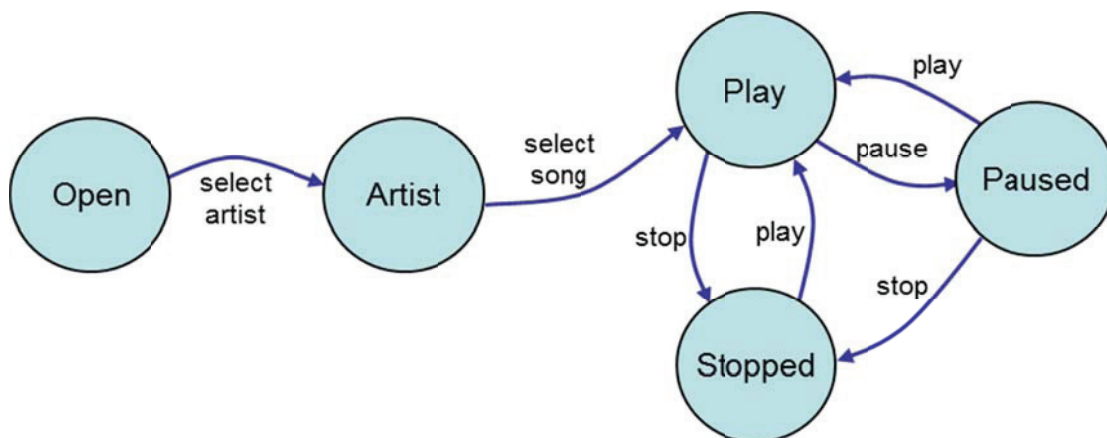
1. Select Artist
2. Select Song
3. Pause
4. Stop
5. Play
6. Select Artist

What State would the application then be in?

- A** Artist
B Play
C Paused
D Stopped

K3 (1 mark)

- 11 The State Transition diagram below reflects a music application on a Smartphone.



Which of the following test cases achieves 100% transition coverage?

- A Select Artist - Select Song - Pause - Stop - Play
- B Select Artist - Select Song - Pause - Stop - Play - Pause - Stop
- C Select Artist - Select Song - Pause - Play - Pause - Stop - Play - Stop
- D Select Artist - Select Song - Pause - Play - Pause - Stop - Play - Pause

K3 (1 mark)

- 12 Given the State Table below, which of the following options represents an invalid transition?

State	Events					
	A	B	C	D	E	F
S1	S2					
S2		S1	S2	S3		
S3					S4	
S4	S1					S4

- A Event B from S2
- B Event D from S3
- C Event C from S2
- D Event A from S4

K3 (1 mark)

- 13 Consider the following testing objectives:

- i. To ensure that every screen-to-screen change in the screen dialogue diagram has been tested.
- ii. To test all combinations of input conditions for high risk complex business rules.

Consider the following test design techniques:

- p) Use Case testing
- q) State transition testing
- r) Boundary value analysis
- s) Decision testing
- t) Decision table testing

Which of the following options shows the best techniques to be used for each objective?

- A i -p; ii -t
- B i -p; ii -s
- C i -q; ii- t
- D i -t, ii -r

K2 (1 mark)

- 14** Assuming that your project is to develop Use Cases as a way of capturing system requirements.

Which of the following test objectives would be best satisfied by employing the Use Case test design technique?

- A To identify defects in component interaction for real world scenarios.
- B To identify defects in components that perform complex business rules.
- C To identify defects in how components handle input field range values.
- D To identify defects when components trigger screen-dialogue flow changes.

K2 (1 mark)

- 15** Which one of the following statements BEST describes why structure based test design techniques are used in component testing?

- A To validate whether a component has adequately met its specification.
- B To locate defects that are hard to find during dynamic testing.
- C To assess the extent of code exercised by a suite of tests.
- D To automate the repetitive element of component test design.

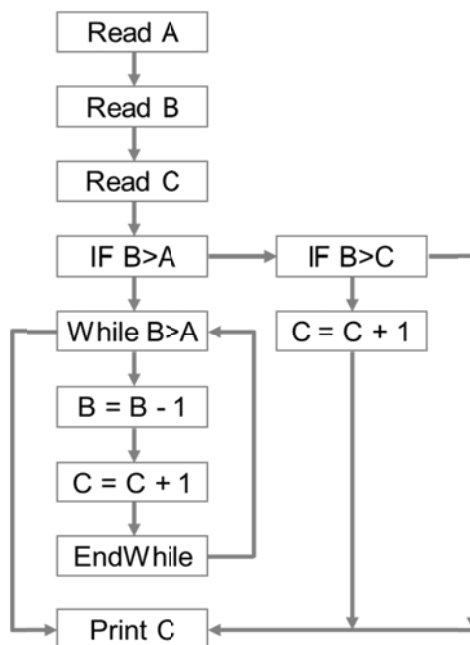
K2 (1 mark)

- 16** Which of the following statements is true for Decision testing?

- A It can be used to assess the percentage of executable statements exercised by a suite of component tests.
- B It is a form of control flow testing that can be used at the component, integration and system test levels.
- C It tests combinations of conditions for complex business rules that a system is to implement.
- D It usually requires fewer test cases than statement testing to achieve 100% coverage of a component's code.

K2 (1 mark)

- 17 Considering the control-flow diagram below, what are the minimum number of tests required to achieve 100% statement coverage (SC) and 100% decision coverage (DC)?



- A SC = 2; DC = 3
 B SC = 3; DC = 3
 C SC = 3; DC = 4
 D SC = 4; DC = 4

K3 (1 mark)

- 18 Considering the pseudo code below and the test suite given in the table below the pseudo-code:

```

1  Read HOME_TEAM_SCORE
2  Read AWAY_TEAM_SCORE
3  IF HOME_TEAM_SCORE > AWAY_TEAM_SCORE
4      PRINT "HOME WIN"
5      HOME_TEAM_POINTS = 3
6      AWAY_TEAM_POINTS = 0
7      IF HOME_TEAM_SCORE > 2* AWAY_TEAM_SCORE
8          HOME_TEAM_POINTS = HOME_TEAM_POINTS + 1
9      ENDIF
10 ELSE
11     IF HOME_TEAM_SCORE = AWAY_TEAM_SCORE
12         PRINT "DRAW"
13         HOME_TEAM_POINTS = 1
14         AWAY_TEAM_POINTS = 1
15     ELSE
16         PRINT "AWAY WIN"
17         HOME_TEAM_POINTS = 0
18         AWAY_TEAM_POINTS = 3
19         IF AWAY_TEAM_SCORE > 2* HOME_TEAM_SCORE
20             AWAY_TEAM_POINTS = AWAY_TEAM_POINTS + 1
21         ENDIF
22     ENDIF
  
```


23 **ENDIF**

Test	HOME_TEAM_POINTS	AWAY_TEAM_POINTS
1	2	3
2	4	1
3	2	2
4	1	2
5	3	0

What percentage of decision coverage has been achieved by this test suite?

- A** 50%
- B** 60%
- C** 75%
- D** 100%

K4 (1 mark)

19 Which of the following describes why experienced-based testing may be of benefit to a project?

- A** To derive additional tests based on analysis of specifications.
- B** To identify special tests not easily captured by formal techniques.
- C** To find important defects before more formal tests are run.
- D** To provide an effective alternative to formal test design techniques.

K1 (1 mark)

20 Which of the following describe the use of experience-based, rather than specification-based techniques?

- i. Creating a list of possible defects and then designing tests to attack these defects.
- ii. Deriving tests systematically from models that specify the system under test.
- iii. Selecting test conditions based on an analysis of the test basis documentation.
- iv. Concurrently designing, executing, logging tests and learning based on a test charter.

- A** i, ii
- B** iii, iv
- C** i, iv
- D** ii, iii

K2 (1 mark)

21 You are testing a high-risk system that calculates an individual's tax liability. The rules that apply for the calculation are complex and involve many conditions. The overall liability must also take into account current legislative tax bands, each reflecting a numerical portion of the individual's taxable income. The developers have completed component testing and you must decide which test design techniques to use for system testing.

Which TWO of the following test design techniques would be MOST suitable?

- i. Decision table testing
- ii. Multiple condition testing
- iii. Boundary value analysis
- iv. Equivalence partitioning
- v. Exploratory testing

- A** i, iii
- B** ii, iii
- C** i, iv
- D** iv, v

K2 (1 mark)

22 Given the code below, which of the following is true?

```

1      Read P
2      Read Q
3      IF P+Q > 100 THEN
4          Print "Large"
5      ELSE
6          IF P+Q > 50 THEN
7              Print "Medium"
8          ELSE
9              Print "Small"
10         ENDIF
11     ENDIF

```

- A** 1 test for statement coverage, 3 for decision coverage.
- B** 2 tests for statement coverage, 3 for decision coverage.
- C** 3 tests for statement coverage, 3 for decision coverage.
- D** 3 tests for statement coverage, 2 for decision coverage.

K3 (1 mark)

23 Given the pseudo code below, which of the following is true?

```

1      Switch on PC
2      Start "outlook"
3      IF emails are received THEN
4          Read the emails
5      ENDIF
6      Close outlook
7      Switch off PC

```

- A** 1 test for statement coverage, 1 for decision coverage.
- B** 1 test for statement coverage, 2 for decision coverage.
- C** 1 test for statement coverage, 3 for decision coverage.
- D** 2 tests for statement coverage, 2 for decision coverage.

K3 (1 mark)

24 Given the pseudo code below, which of the following is true?

```

1      IF A > B THEN
2          C = A - B
3      ELSE
4          C = A + B
5      ENDIF
6      Read D
7      IF C = D Then
8          Print "Error"
9      ENDIF

```

- A** 2 tests for statement coverage, 2 for decision coverage.
- B** 2 tests for statement coverage, 3 for decision coverage.
- C** 3 tests for statement coverage, 3 for decision coverage.
- D** 3 tests for statement coverage, 2 for decision coverage.

K3 (1 mark)

- 25** Given the pseudo code below, which of the following is true (where SC means minimum number of tests for statement coverage and DC means minimum number of tests for decision coverage)?

```

1      Read A
2      Read B
3      IF B = A THEN
4          Print "they are the same"
5      ELSE
6          Print "they are different"
7      ENDIF

```

- A** SC = 1, DC = 1.
- B** SC = 1, DC = 2.
- C** SC = 2, DC = 1.
- D** SC = 2, DC = 2.

K3 (1 mark)

- 26** Given the procedure below, which of the following is true (where SC means minimum number of tests for statement coverage and DC means minimum number of tests for decision coverage)?

```

1      Go to the vending machine
2      If the vending machine is not working then
3          call repair centre to fix
4      Otherwise
5          Insert money
6          If enough money is inserted then
7              Select a drink
8              Wait for drink to be dispensed
9          Otherwise
10         Display message requesting more money

```

- A** SC = 2, DC = 2.
- B** SC = 3, DC = 3.
- C** SC = 2, DC = 3.
- D** SC = 3, DC = 2.

K3 (1 mark)

- 27** Given the pseudo code below, which of the following is true (where SC means minimum number of tests for statement coverage and DC means minimum number of tests for decision coverage)?

```

1      Questions = 40
2      Result = (Right / Questions) * 100
3      IF Result < 61 THEN
4          print "FAIL"
5          X = 0
6      ELSE

```

```
7      print "PASS"
8      X = 1
9      ENDIF
10     IF X = 0 THEN
11         Print "You may retake the exam"
12     ELSE
13         Print "Congratulations!"
14     ENDIF
```

- A** SC = 2, DC = 3.
- B** SC = 3, DC = 3.
- C** SC = 2, DC = 2.
- D** SC = 2, DC = 4.

K3 (1 mark)

CTFL-4.1.1 K2 Differentiate between a test design specification, test case specification and test procedure specification.

1 The answer is B.

Per syllabus: 'During test analysis, the test basis documentation is analysed in order to determine what to test, i.e. to identify the test conditions' (also known as test items).

IEEE 829 defines the Test Design document as containing the test items 'Identify the test items and describe the features and combinations of features that are the object of this design specification'.

CTFL-4.1.2 K2 Compare the terms test condition, test case and test procedure.

2 The answer is C.

- A NO. These are defined in the test plan.
- B NO. These are defined in a test procedure.
- C YES. Per syllabus 'A test case consists of a set of input values, execution pre-conditions, expected results and execution post-conditions, developed to cover a certain test objective(s) or test condition(s)'.
- D NO. This is defined in a test execution schedule.

CTFL-4.1.3 K2 Evaluate the quality of test cases in terms of clear traceability to the requirements and expected results.

3 The answer is D.

Per syllabus: 'Traceability from test conditions (and subsequently the test cases that are linked to the test conditions) back to the specifications and requirements enables both effective impact analysis when requirements change ...'.
Therefore (i).

Per syllabus: 'If expected results have not been defined, then a plausible, but erroneous, result may be interpreted as the correct one'.
Therefore (iv).

CTFL-4.1.4 K3 Translate test cases into a well-structured test procedure specification at a level of detail relevant to the knowledge of the testers.

4 The answer is B.

The correct test procedure must have the sequence of events correct, must test for both valid and invalid entries (to meet objectives of the test case) and must provide enough details for an inexperienced tester.

- A NO. Sequence correct; level of detail correct; **not testing invalid payment entry.**
- B YES. Sequence correct, level of detail correct; tests both valid and invalid payment entry.
- C NO. Sequence correct, **level of detail insufficient (for valid and invalid entries);** tests both valid and invalid payment entry.

- D NO. **Sequence incorrect (recipient should be first action)**, level of detail correct; tests both valid and invalid payment entry.

CTFL-4.2.1 K1 Recall reasons that both specification-based (black-box) and structure-based (white-box) test design techniques are useful and list the common techniques for each.

5 The answer is C.

We are looking for a white-box technique (structure based). Per syllabus 'White-box test design techniques (also called structural or structure-based techniques) are based on an analysis of the structure of the component or system'. Options A, B and D are all black-box techniques.

CTFL-4.2.2 K2 Explain the characteristics, commonalities, and differences between specification based testing, structure-based testing and experience-based testing.

6 The answer is B.

- A NO. Use Case testing is a specification based technique. There are no useful specifications.
- B YES. Per syllabus, 'It is an approach that is most useful where there are few or inadequate specifications and severe time pressure'. Given the we are told there is little time for test design, exploratory testing is the best option.
- C NO. This is a structure based technique. You need a black-box technique for functional testing.
- D NO - this is a test approach that requires product risks to be identified and assessed before tests can be designed. We are told there is little time for test design and nothing is said of risk identification and analysis having been done, so it seems likely that this approach may need more time than there is available.

CTFL-4.3.1 K3 Write test cases from given software models using **equivalence partitioning**, boundary value analysis, decision tables and state transition diagrams/tables.

7 The answer is C.

1 st Jan	28 th Feb	1 st Mar	31 st May	1 st Jun	31 st Aug	1 st Sep	30 th Nov	1 st Dec	31 st Dec
Closed (Invalid)		Spring (Valid)		Summer (Valid)		Autumn (Valid)		Closed (Invalid)	

The equivalence partition model for the scenario is shown above. Note that there are 2 invalid partitions, each representing segments at either end of the winter season.

To apply EP correctly there should be one test case (and one only) in each valid and invalid partition. So:

- A NO. Misses a test cases for the (valid) summer season partition
- B YES. One test case for each of the 5 partitions (3 valid and 2 invalid)
- C NO. This is applying BVA and therefore has unnecessary test cases for each partition as far as EP is concerned.
- D NO. Covers the 3 valid partitions only.

CTFL-4.3.1 K3 Write test cases from given software models using equivalence partitioning, **boundary value analysis**, decision tables and state transition diagrams/tables.

8 The answer is C.

1 st Jan	28 th Feb	1 st Mar	31 st May	1 st Jun	31 st Aug	1 st Sep	30 th Nov	1 st Dec	31 st Dec
Closed (Invalid)		Spring (Valid)		Summer (Valid)		Autumn (Valid)		Closed (Invalid)	

The equivalence partition model for the scenario is shown above. Note that there are 2 invalid partitions, each representing segments at either end of the winter season.

To apply (the 2 value) BVA technique correctly there should be one test case representing a value either side of each partition boundary, i.e. the minimum and maximum boundary values shown in the diagram. However, it is important to note that the question asks for the valid boundary values only, so we must discount the values in the 2 invalid partitions.

- A NO. Incorrectly covers the invalid boundary values as well.
- B NO. 30th May; 2nd June; 30th August; 2nd September are not BVA values as they are more than one increment away from the boundaries.
- C YES. Covers all 6 valid BVA values.
- D NO. The increments must be in days as it is a 365 day calendar, not whole months.

CTFL-4.3.1 K3 Write test cases from given software models using equivalence partitioning, boundary value analysis, **decision tables** and state transition diagrams/tables.

9 The answer is D.

TC1 Roger - falls into Rule 4, so a £10 fixed charge
 TC2 Carol - falls into Rule 1 (has a Gold card which overrides all other conditions), so no extra charge.

So option D.

CTFL-4.3.1 K3	Write test cases from given software models using equivalence partitioning, boundary value analysis, decision tables and state transition diagrams/tables .
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------

10 The answer is B.

The key to solving this question is recognising that there can be invalid transitions which these diagrams do not easily show (State transition tables would do).

The last event '6. Select Artist' is an invalid transition, as that event can only cause a transition when currently in the Open state, whereas after event 5 in the sequence the application would be in the Play state. So it would remain in the Play state.

CTFL-4.3.1 K3	Write test cases from given software models using equivalence partitioning, boundary value analysis, decision tables and state transition diagrams/tables .
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------

11 The answer is C.

Only option C covers all 7 transitions shown in the diagram, which is what is required to achieve 100% transition coverage.

Option A covers all 5 states but not all transitions.

Option B misses the Paused to Play transition.

Option D misses the Play to Stopped transition.

CTFL-4.3.1 K3	Write test cases from given software models using equivalence partitioning, boundary value analysis, decision tables and state transition diagrams/tables .
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------

12 The answer is B.

Option B - whilst currently in State S3, the only event that will cause a valid transition is E (transitioning to S4). D from S3 shows an empty cell (an invalid transition).

CTFL-4.3.2 K2	Explain the main purpose of each of the four testing techniques, what level and type of testing could use the technique, and how coverage may be measured.
---------------	------------------------------------------------------------------------------------------------------------------------------------------------------------

13 The answer is C.

- i. (q) State transition testing - per syllabus 'However, the technique is also suitable for modelling a business object having specific states or testing screen-dialogue flows (e.g., for Internet applications or business scenarios)'. transition coverage would achieve the objective.
- ii. (t) Decision Table testing - per syllabus 'Decision tables may be used to record complex business rules that a system is to implement. The coverage standard commonly used with decision table testing is to have at least one test per column in the table, which typically involves covering all combinations of triggering conditions.'

14 The answer is A.

- A TRUE - Per syllabus 'Test cases derived from use cases are most useful in uncovering defects in the process flows during real-world use of the system. They also help uncover integration defects caused by the interaction and interference of different components, which individual component testing would not see'.
- B FALSE - Best to use decision table testing.
- C FALSE - Best to use boundary value analysis.
- D FALSE - Best to use state transition testing.

15 The answer is C.

- A NO - This is specification (black-box) testing.
- B NO - This describes static testing (static analysis).
- C YES - e.g. 'statement coverage is the assessment of the percentage of executable statements that have been exercised by a test case suite'.
- D NO - Describes a benefit of tool support.

16 The answer is B.

- A NO - This describes statement coverage.
- B YES - Per syllabus 'Decision testing is a form of control flow testing as it follows a specific flow of control through the decision points' and 'The concept of (structure-based) coverage can also be applied at other test levels' such as integration and system test levels.
- C NO - Describes decision table testing.
- D NO - The inverse is true.

17 The answer is A.

- 2 tests are needed for 100% statement coverage.
- 3 tests are needed for 100% decision coverage.

There needs to be one more Decision test than Statement test to cover the True branch from the 'IF B > C' statement - Statement testing does not need to cover this branch - So options B and D can be eliminated straight away.

For the While loop we only need one test (initially making the While statement true) to achieve 100% decision coverage, as the code should also make it false eventually (unless it is an infinite loop!). So 1 test is needed when B > A, not 2 as would be required for an IF statement. Hence A is the correct answer, not B.

CTFL-4.4.4 K4	Assess statement and decision coverage for completeness with respect to defined exit criteria.
---------------	------------------------------------------------------------------------------------------------

18 The answer is C.

To assess decision coverage, you must count the number of true and false branches in the code structure (in this case 8 as there are 4 ifs at lines 3, 7, 11 and 19) and assess how many of these the suite of tests have covered.

These 5 test cases only cover 6 of the 8 branches (hence 75% coverage achieved). They miss the False of the second IF at line 7 (i.e. where HOME > AWAY but not > 2*) and the True of the last IF at line 19 (i.e. where AWAY > HOME but is > 2*).

CTFL-4.5.1 K1	Recall reasons for writing test cases based on intuition, experience and knowledge about common defects.
---------------	----------------------------------------------------------------------------------------------------------

19 The answer is B.

- A NO - this relates to specification-based techniques. Experience-based testing does not require analysis of specifications.
- B YES - Per syllabus 'these techniques can be useful in identifying special tests not easily captured by formal techniques ...'.
- C NO - They should be applied after more formal techniques, not before.
- D NO - They should complement more formal techniques, not replace them.

CTFL-4.5.2 K2	Compare experience-based techniques with specification-based testing techniques.
---------------	----------------------------------------------------------------------------------

20 The answer is C.

- i. YES - Describes a fault attack (error guessing).
- ii. NO - specification based testing: 'Models, either formal or informal, are used for the specification of the problem to be solved. Test cases can be derived systematically from these models'.
- iii. NO - specification based testing: 'Specification-based techniques are a way to derive and select test conditions, test cases, or test data based on an analysis of the test basis documentation'.
- iv. YES - Describes exploratory testing.

CTFL-4.6.1 K2	Classify test design techniques according to their fitness to a given context, for the test basis, respective models and software characteristics.
---------------	----------------------------------------------------------------------------------------------------------------------------------------------------

21 The answer is A.

- i. YES - Decision table testing is ideal where there are complex business rules with triggering conditions.
- ii. NO - Sounds tempting (with the mention of many conditions) but, as a white-box technique, would normally be used during component testing.
- iii. YES - BVA is ideal where there are numerical ranges which would have boundary values, such as tax bands.
- iv. NO - Could be used for the tax bands but BVA would be stronger, as this is a high-risk system.

- v. NO - Could be used to augment the other techniques but not one of the best 2 techniques. Possibly a third choice.

CTFL-4.4.3 K3 Write test cases from given control flows using statement and decision test design techniques.

22 The answer is C.

For statement coverage we need one test to exercise the true outcome of the decision statement at line 3 so it then executes the statements at lines 4 and 5, giving us a path of (1 – 5, 11). A second test case should then take the false outcome of the decision statement at line 3. This will take us to the line 6. If this second test case then has a true outcome for the decision statement at line 6, it will cover the path (1 – 3, 6 – 8, 10, 11).

This leaves just line 9 not yet covered. To cover this we will need a third test case that also has a false outcome for the decision statement at line 3 and a false outcome for the decision statement at line 6. This will cover the path (1 – 3, 6, 9 – 11).

Having achieved 100% statement coverage with three test cases we can see that these same test cases have also achieved 100% decision coverage since they have exercised both the true and false outcomes of both the decision statements.

CTFL-4.4.3 K3 Write test cases from given control flows using statement and decision test design techniques.

23 The answer is B.

If we have a test that results in a true outcome for the decision statement at line 3 (i.e. emails are received) then that test case will exercise all the statements from line 1 through to line 7 inclusive. So 1 test is sufficient to achieve 100% statement coverage.

For 100% decision coverage we will also need to exercise the false outcome for the decision statement at line 3 (i.e. no emails are received), so a second test will be necessary.

CTFL-4.4.3 K3 Write test cases from given control flows using statement and decision test design techniques.

24 The answer is A.

For 100% statement coverage we will need to exercise the true outcome of the decision at line 1 (to cover the statements at lines 2 and 3) and the false outcome (to cover the statement at line 4). This will require 2 tests. Providing at least one of the tests also exercises the true outcome for the decision at line 6, then all the statements can be covered with just two tests.

For 100% decision coverage just two tests will be sufficient to exercise the true and false outcomes of both the decision statements.

25 The answer is D.

Two tests are required to achieve 100% statement coverage and 100% decision coverage.

26 The answer is B.

To cover the procedural step at line 3 we need a true outcome for the decision at line 2. A test that does this will cover the path (1 – 4). Note that there is no equivalent to an ENDIF statement in this procedure.

To get to line 5 we need a false outcome for the decision at line 2. A test that does this will go on to exercise the decision statement at line 6. If this has a true outcome the same test will then cover the statements at lines 7 through 10 inclusive.

So 2 tests are needed for 100% statement coverage. However, these two tests only achieve 75% decision coverage as a false outcome for the decision statement at line 6 has not been exercised. So a third test is needed for 100% decision coverage.

27 The answer is C.

For 100% statement coverage we need a true outcome for the decision statement at line 3 to cover statements at lines 4 to 6. We need a false outcome for the decision at line 3 to cover statements at lines 7 and 8. This will need 2 tests. These same tests can be used to cover the statements from 10 to 14: a true outcome at line 10 will cover lines 11 and 12, a false outcome will cover line 13.

Practice Exam 5_2 Questions

1 Which of the following explains why independent testing is important?

- A Because independent testers are more concerned for quality than developers.
- B Because independent testers find defects faster than developers.
- C Independent testing allows developers to focus all their effort on writing code.
- D Because independent testers will find defects missed by developers.

K1 (1 mark)

2 Which of the following explains both the benefits and drawbacks of independent testing?

- A An independent tester can verify assumptions people made during specification and implementation of the system but can become isolated from the development team.
- B Independent testers see other and different defects as they are not biased towards the product but can lose a sense of responsibility for quality.
- C Independent testers find more defects when remaining totally independent from the development team but can be biased towards the requirements.
- D Independent testers always make correct assumptions about the specifications but may be seen as a bottleneck or blamed for delays in release.

K2 (1 mark)

3 Which of the following roles would NOT normally be considered for a testing role on a project?

- A Developer
- B System operator
- C Auditor
- D Certification specialist

K1 (1 mark)

4 Which of the following would normally be performed by a test leader rather than a tester?

- A Analyse, review and assess user requirements and models for testability.
- B Write test summary reports based on the information gathered during testing.
- C Measure performance of components and systems and create performance specifications.
- D Review and contribute to test plans and review tests developed by others.

K1 (1 mark)

5 Which one of the following statements is TRUE for test planning?

- A The test policy and level test plans should be influenced by the objectives defined in the master test plan.
- B Planning may be documented in a master test plan and in separate level test plans with objectives defined in each document.
- C There may be a master test plan and level test plans for a project but only the test policy defines test objectives.
- D A master test plan is written at the beginning of a project and must not change during the remainder of the test lifecycle.

K1 (1 mark)

- 6 Which of following items would you NOT expect to find in a master test plan, according to IEEE 829?
- i. The principles, approach and major objectives of the organization regarding testing.
 - ii. The necessary and desired properties of the test environments.
 - iii. The overall approach for testing for each major group of features.
 - iv. Staffing needs by skill level and training options for providing necessary skills.
 - v. The priorities of the test conditions following analysis of the test basis .
- A** i and ii
B ii and iii
C iv and v
D i and v

K2 (1 mark)

- 7 A master test plan has determined that both risk-based and requirements-based testing should be performed. Which of the following test approaches does this reflect?
- A** A methodical approach.
B An analytical approach.
C A process-compliant approach.
D A dynamic approach.

K2 (1 mark)

- 8 Which of the following statements is true regarding test execution?
- A** The test leader determines when test execution starts and records this in the test execution schedule.
B The order in which the test procedures will be run is added to the test plan by the tester.
C The test execution schedule is normally produced by the tester and is influenced by the test plan.
D The test leader schedules test execution as soon as test analysis and design is complete and adds this to the test plan

K2 (1 mark)

- 9 You have been asked to develop a test execution schedule for a bug-fix release of a shopping cart application.

The application allows the customer, for each new order, to perform the following functions, shown in logical order of processing (Steps 1 to 4):

Step 1 - Add new and/or remove existing items to/from the cart.

Step 2 - Checkout.

Step 3 - Confirm existing card details / Add new card details.

Step 4 - Cancel Order or Confirm order.

The test plan specifies that only bug-fix test procedures should be run in the release. The following test procedures are available for consideration:

Test procedure		Open defect?
p	Checkout	Y
q	Remove existing item from cart	Y
r	Cancel Order	Y
s	Add new card details	N

t	Add new item to cart	N
u	Confirm existing card-details	Y
v	Confirm Order	N

Assuming that a new order already exists with no items currently added to it, which would be the correct test execution sequence for the release?

- A** t, q, t, p, u, v, r
- B** t, q, t, p, u, r
- C** t, q, p, u, r
- D** q, p, u, r

K3 (1 mark)

10 Which one of the following is a test preparation and execution activity that applies during the test planning stage?

- A** Setting the level of detail for test procedures to support reproducible test preparation and execution.
- B** Analysing and evaluating the testability of the test basis and test objects.
- C** Verifying and updating bi-directional traceability between the test basis and test cases.
- D** Providing feedback and visibility about test execution by way of metrics.

K1 (1 mark)

11 Which one of the following has the GREATEST influence on estimating the effort required for testing?

- A** The number of lines of code that need to be developed.
- B** The number of testers that are currently available.
- C** The number of defects and the amount of rework required.
- D** The number of test cases that have been written.

K1 (1 mark)

12 Which one of the following is TRUE regarding test estimation approaches?

- A** A metrics-based approach will consistently produce a better quality estimate than an expert-based approach.
- B** An expert-based approach does not require historical metrics whereas a metrics-based approach can be based on similar projects.
- C** An expert-based approach is better for Agile projects whereas a metrics-based approach is better for Sequential development models.
- D** All projects must apply both expert-based and metrics-based approaches and then average the two for the best estimate.

K2 (1 mark)

13 Which one of the following is a good example of entry and exit criteria for system test execution?

- A** Entry - Testers available to run tests;
Exit - pass rate for component integration testing met.
- B** Entry - Test automation tools ready for use;
Exit - Test automation scripts complete.
- C** Entry - Test basis reviewed and signed-off;
Exit - Open defects below agreed thresholds.
- D** Entry - Test environments built and verified;
Exit - product risks successfully tested.

K2 (1 mark)

14 Which one of the following is NOT a metric that could be used to assess progress towards meeting exit criteria for test execution?

- A** Testing budget expended
- B** Defects found and fixed
- C** Test cases automated
- D** Tests executed and passed

K2 (1 mark)

15 Which one of the following describes a test metric that suggests test control should be initiated?

- A** Static analysis indicates that the code structure is too complex.
- B** 40% of the performance tests have failed test execution.
- C** The number of test cases written equals 70% of the planned figure.
- D** There are three high severity defects open after the first test cycle.

K2 (1 mark)

16 Which of the following would you NOT expect to find in a test summary report?

- A** Recommended process improvements based on lessons learned.
- B** A risk assessment for defects that still remain open.
- C** Instances of where the test plan was not followed and why.
- D** Whether it is financially beneficial to extend testing beyond the planned date.

K2 (1 mark)

17 Which of the following best describes how configuration management supports testing?

- A** Configuration management maintains product integrity throughout the project but not after the product is released into production.
- B** Configuration management prevents developers accessing the system tests while allowing testers free access to all testware.
- C** Configuration management help testers uniquely identify testware including test harnesses.
- D** Configuration management determines the configuration management procedures that test planning should use.

K2 (1 mark)

18 A project is starting to update a company website. The two stakeholders are the Customer Support Director and the Sales Director. The two objectives for the website project are:

- To reduce direct calls to the Support Line for UK customers, by allowing customers to access “self-service” help instead of contacting Customer Support directly.
- To reduce direct calls to the Sales Office by allowing customers to purchase products directly from the website.

Which of the following is a risk for this website project?

- A** If the on-line shop is not functional but the website increases demand for the product, the number of calls to the Sales Office might increase.
- B** The project costs might be high so that the project does not result in an increased profit for the company even if sales increase.
- C** Potential customers in France (a large market) and Japan (a growing market) may not be able to use English language website Support FAQs.
- D** The Despatch and Delivery Manager is concerned that his team will be overwhelmed with additional orders, causing delivery times to be unacceptable.

K2 (1 mark)

19 Which of the following statements provides us with the information we need to determine the level of risk for a software failure?

- A** The impact of the failure each time it occurs is 2 hours downtime with a loss of 10,000 USD per hour, but the likelihood of it happening is low.
- B** There is a high likelihood of the failure happening once a week under normal operating conditions, but a low likelihood of it happening more frequently.
- C** The impacts of the software failure include having to halt normal business operation for up to 2 hours per failure and some commercial damage.
- D** The team is uneasy about software quality based on previous experience of defect levels in the code being changed.

K1 (1 mark)

20 You are asked to identify significant project and product risks for your testing project. Which of the following are project risks and which are product risks?

- i. The software might fail frequently when customers use it.
- ii. The test environment might not be ready in time for testing.
- iii. There may be data corruption following the data migration.
- iv. Some people may be moved to another testing project.
- v. The system may not be usable for its intended audience.

- A** i, iii and v are Product Risks, ii and iv are Project Risks
- B** i, ii, iii and iv are Product Risks, v is a Project Risk
- C** i is a Product Risk, ii, iii, iv and v are Project Risks
- D** i and v are Product Risks, ii, iii, and iv are Project Risks

K2 (1 mark)

21 Your project manager has asked you to identify significant risks for a system testing project. Which one of these is a PROJECT risk?

- A** Low quality code is delivered into system test.
- B** Failure prone software is delivered to acceptance test.
- C** The software does not deliver the required functionality.
- D** The software has a long response time.

K1 (1 mark)

22 A project is starting to update a company website. The two stakeholders are the Customer Support Director and the Sales Director. The two objectives for the website project are:

- To reduce direct calls to the Support Line for UK customers, by allowing customers to access “self-service” help instead of contacting Customer Support directly.
- To reduce direct calls to the Sales Office by allowing customers to purchase products directly from the website.

The following are among the risks that have been identified by the test team:

- i. The online price list may be wrongly entered to the website shop during the data migration.
- ii. The Customer Support Director and Sales Director may find it difficult to agree one set of non-conflicting requirements.
- iii. The third party supplier for the on-line shop may fail to deliver on time.
- iv. The website is prone to failures and crashes, including failures of the checkout process.
- v. The support area in the website has a long response time, meaning potential customers “click away”.
- vi. The conversion of the Support FAQs from the Support in-house system to a format for the website might be late.

Which of these risks will be considered by the Test Manager during test planning for a risk-based approach?

- A** Risks i, iv and v directly affect the test planning.
- B** Risks ii, iii, and vi directly affect the test planning.
- C** only Risk iv directly affects the test planning.
- D** only Risk ii directly affects the test planning.

K2 (1 mark)

23 Which of the following is NOT part of a test incident report as described in IEEE Std 829-1998?

- A** Item pass/fail criteria
- B** Expected and actual results
- C** Environment for the test
- D** Impact of the incident

K1 (1 mark)

24 The following is part of a test incident report that is among those raised by the test team for a company website:

- Test Incident Report Identifier: TIR23
- Summary: An item that should be priced with a 10% discount if you order more than 5 applies the discount if you order 5 items.

- Incident Description
 - Input: 5 dustbins at 10.99 each
 - Expected results: no discount applied
 - Actual results: 10% discount applied
- (test incident continues...)

Which of the following would be most likely to improve the test incident report?

- A** Including the expected and actual results for other values (4, 6) to show whether the software works for fewer than 5 items or for more than 5 items.
- B** Including all the key steps required to reproduce the problem, in a detailed procedure specification.
- C** Including more detail of the calculation done and exact pricing to get the expected and actual results.
- D** Including details of the test design method used to produce this test.

K3 (1 mark)

CTFL-5.1.1 K1 Recognize the importance of independent testing.

1 The answer is D.

- A NO - Unfair statement - both should be concerned for product quality.
- B NO - It is not about speed.
- C NO - Developers also have testing responsibilities (e.g. component testing, static analysis).
- D YES - 'Independent testers see other and different defects, and are unbiased'.

CTFL-5.1.2 K2 Explain the benefits and drawbacks of independent testing within an organization.

2 The answer is A.

- A TRUE and TRUE per syllabus.
- B TRUE - FALSE (applies to developers).
- C FALSE (this can result in a drawback) - FALSE (the opposite).
- D FALSE (they can verify assumptions made by others but do not always make correct assumptions themselves) – TRUE.

CTFL-5.1.3 K1 Recognize the different team members to be considered for the creation of a test team.

3 The answer is C.

- A YES - e.g. Component testing.
- B YES - e.g. Operational Acceptance testing.
- C NO - may audit testing but would not perform it.
- D YES - 'certification testers (who certify a software product against standards and regulations)'.

CTFL-5.1.4 K1 Recall the tasks of typical test leader and tester

4 The answer is B.

- A NO - Tester (this is a common mistake made about the roles, from my experience as a trainer).
- B YES - exactly per syllabus.
- C NO - both done by testers.
- D NO - both done by testers (again a common mistake).

CTFL-5.2.1 K1 Recognize the different levels and objectives of test planning.

5 The answer is B.

- A FALSE - Inverse is true. The test policy influences all beneath it (test strategy, then master test plans then level test plans).
- B TRUE - as per syllabus.
- C FALSE - There should test objectives within all test plans.
- D FALSE - 'Test planning is a continuous activity and is performed in all life cycle processes and activities'.

CTFL-5.2.2 K2	Summarize the purpose and content of the test plan, test design specification and test procedure documents according to the 'Standard for Software Test Documentation' (IEEE Std 829-1998).
---------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6 The answer is D.

- NO - this is the Test Policy (word-for-word per Glossary of Terms).
- YES - per IEEE 829.
- YES - per IEEE 829.
- YES - per IEEE 829.
- NO - This is performed during test analysis and design and recorded in the test design specification.

CTFL-5.2.3 K2	Differentiate between conceptually different test approaches, such as analytical, model-based, methodical, process/standard compliant, dynamic/heuristic, consultative and regression-averse.
---------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7 The answer is B.

- B - Per Syllabus 'Analytical approaches, such as risk-based testing where testing is directed to areas of greatest risk'.

CTFL-5.2.4 K2	Differentiate between the subject of test planning for a system and scheduling test execution.
---------------	------------------------------------------------------------------------------------------------

8 The answer is C.

This questions hinges on understanding the difference between scheduling test execution (which is performed by the test leader during test planning) and producing a test execution schedule which is undertaken by the tester during test implementation.

- A FALSE - This is recorded in the test plan.
- B FALSE - This is recorded in the test execution schedule.
- C TRUE - The test execution schedule will be driven by the key dates for test execution defined in the test plan.
- D FALSE - scheduling test execution should be done before test analysis and design starts as it will influence test estimates and resource levels.

CTFL-5.2.5 K3	Write a test execution schedule for a given set of test cases, considering prioritization, and technical and logical dependencies.
---------------	------------------------------------------------------------------------------------------------------------------------------------

9 The answer is B.

Option B: Add new item to cart (t) must be run first even though it is not a bug-fix procedure (i.e. not in scope), as without it there would be no order to process. Then we can get to Remove existing item from cart (q) which is in scope. Next in sequence would have to be (t) again in order to have an item in the order (we removed the last one). Then we can Checkout (p) which is in scope, and Confirm existing card details (u) also in scope, then finally we Cancel Order (r) in scope.

Option A incorrectly adds Confirm Order which is not in scope and can't be done as well as Cancel.

Option C fails to re-run Add new item to cart so there would be nothing to checkout.

Option D just picks the four procedures that are in scope and places them in a seemingly logical order - but this won't work as no new items would exist to checkout the (empty) order.

CTFL-5.2.6 K1 List test preparation and execution activities that should be considered during test planning.

10 The answer is A.

- A YES - exactly per syllabus 5.2.2 'Setting the level of detail for test procedures in order to provide enough information to support reproducible test preparation and execution'.
- B NO - Part of Test Analysis.
- C NO - Part of Test Implementation.
- D NO - This is Test Progress Monitoring.

CTFL-5.2.7 K1 Recall typical factors that influence the effort related to testing

11 The answer is C.

- A NO - In the absence of better information (metrics or expert advice) we could provisionally measure testing effort as a percentage of development effort, but first we would have to convert the number of lines of code into development effort. Not the best answer.
- B NO - Test estimation should determine how many testers we need based on the product, process and outcome of testing and not based on how many testers we currently have.
- C YES - Per syllabus, this is the outcome of testing.
- D NO - We should be estimating this work!

CTFL-5.2.8 K2 Differentiate between two conceptually different estimation approaches: the metrics based approach and the expert-based approach.

12 The answer is B.

- A NO - Neither approach is better - it all depends on context (available metrics, knowledge, risk etc.).
- B YES - Per syllabus.
- C NO - Both approaches can be used for all development models.
- D NO - Can be advantageous to apply both but not mandated - again each approach depends on what is available.

CTFL-5.2.9 K2	Recognize/justify adequate entry and exit criteria for specific test levels and groups of test cases (e.g., for integration testing, acceptance testing or test cases for usability testing).
---------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

13 The answer is D.

- A NO - Entry OK;
Exit NOT OK - this is exit criteria for the previous test level and thus more likely to be entry criteria for this level.
- B NO - Entry OK;
Exit NOT - this would also be entry criteria.
- C NO - Entry NOT OK - this would apply to test analysis and design, not execution;
EXIT - OK
- D YES for both - examples as per syllabus.

CTFL-5.3.1 K1	Recall common metrics used for monitoring test preparation and execution.
---------------	---------------------------------------------------------------------------

14 The answer is C.

A, B and D are all covered as examples of common test metrics in the bulleted list in section 5.3.1 of the syllabus. C could also be a test metric but unlike the other three, represents an entry criterion for test execution, rather than an exit criterion. See also syllabus section 5.2.4, Exit Criteria.

CTFL-5.3.2 K2	Explain and compare test metrics for test reporting and test control (e.g., defects found and fixed, and tests passed and failed) related to purpose and use.
---------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------

15 The answer is C.

'Test control describes any guiding or corrective actions taken as a result of information and metrics gathered and reported. Actions may cover any test activity and may affect any other software life cycle activity or task'.

If a metric tells us that we are not where we want to be (in terms of schedule, test budget or product quality) then test control must be taken.

- A NO - An issue that requires development team control, not test control.
- B NO - There is nothing to indicate whether 40% is above or below expectation.
- C YES - This indicates that we are not where we should be, so test control is needed.
- D NO - As with B, is this below expectations? We don't know. Perhaps we expected at least this number in the first cycle.

CTFL-5.3.3 K2	Summarize the purpose and content of the test summary report document according to the 'Standard for Software Test Documentation' (IEEE Std 829-1998).
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------

16 The answer is A.

- A NO - this is performed during the final stage of the test process, Test Closure.
- B YES - as per syllabus (such as an assessment of defects remaining and outstanding risks) and IEE 829 1998 (section 11.2.5/6).

- C YES - IEEE 829 1998 (Section 11.2.3 Variances).
- D YES - as per syllabus (the economic benefit of continued testing).

CTFL-5.4.1 K2 Summarise how configuration management supports testing.

17 The answer is C.

- A NO – configuration management maintains product integrity throughout the project and throughout the life of the product (i.e. after the product has been released).
- B NO – configuration management may prevent unauthorised access to testware by anybody (developers, testers, users, etc.) but should be set up to allow appropriate access by all project team members that need the access.
- C YES – being able to uniquely identify individual testware items and relate them to each other and to development items helps to ensure the correct testware items are used in testing and that none are lost.
- D NO – it is the test planning activity that chooses, documents and implements appropriate configuration management procedures.

CTFL-5.5.1 K2 Describe a risk as a possible problem that would threaten the achievement of one or more stakeholders' project objectives.

18 The answer is A.

- A YES – this risk affects the Sales Director (an identified stakeholder) and the objective of reducing calls to the Sales Office.
- B NO – increasing the profit is not identified as an objective. However, in a real project, we would want to ask if this should be an objective or not.
- C NO – the market is identified as UK for this project. However, in a real project we might want to check that, and ask if there is an intention to spread the scope after this project.
- D NO – the Despatch and Delivery Manager is not identified as a stakeholder, but in a real project we would want to ask if he should be one, or to refine this risk to show the risk to sales and support longer term.

CTFL-5.5.2 K1 Remember that the level of risk is determined by likelihood (of happening) and impact (harm resulting if it does happen).

19 The answer is A.

- A YES – a likelihood of the failure event occurring has been estimated on a “high-medium-low” scale and a quantitative assessment of the impact in time and money lost has been made. Combined, these can be used to assess the risk and determine its level.
- B NO – in this answer, two likelihoods of failure have been identified, but no impact. A software failure might be very significant, or trivial. Although both likelihoods are important to understand, unless we know the impact of a software failure we don't know the level of risk.
- C NO – in this answer we identify only some impacts in terms of time lost to the business and possible costs that have not been quantified. The likelihood of the failure has not been identified.
- D NO – in this answer a potential root cause of failure has been identified, based on past experience. An investigation based on defects history and which areas of the code are to be changed, would allow the team to assess

the likelihood of failure, and failure history from previous versions gives the impact of failures from those defects.

CTFL-5.5.3 K2 Distinguish between the project and product risks.

20 The answer is A.

- A** YES – being unable to provide resources of any sort in a timely manner is a project risk, so unavailability of test environments or people should be raised as project risks. Software failure in use, data migration/data corruptions and usability problems are all product risks: they are about our ability to deliver the right product fit for purpose.
- B** NO – being unable to provide resources of any sort in a timely manner is a project risk, so unavailability of people and test environments should be raised as a project risk not a product risk. The product not being usable is a usability (software attribute) risk.
- C** NO – data migration/corruption and usability problems in the delivered product are also product risks, as the product must not only be functionally appropriate but also have the right attributes and supporting data and other artefacts.
- D** NO – data migration/data corruption is also a product risk, because uncorrupted, available data is part of the delivered solution for the customer.

CTFL-5.5.4 K1 Recognise typical project and product risks.

21 The answer is A.

- A** YES – this is a project risk, because if the code is lower quality than we anticipated, the test/refix/retest/regression test activities will take longer than estimated, threatening the project progress and planned completion date.
- B** NO – this is a product risk – if failure prone software is delivered into acceptance test or to production, then this is a product problem, which may have one or more root causes. To counter this risk, we might consider more effort to improve design, coding and testing.
- C** NO – this is a product risk – if the software does not provide the required functionality, then this is a product problem, which may have one or more root causes. To counter this risk, we might consider more effort to improve requirements definition and testing. We might also consider earlier/more continuous involvement of end users/customers in the project.
- D** NO – this is a product risk – if the software has poor performance, for example slow response times, then this is a product problem, which may have one or more root causes. To counter this risk, we might consider more effort to improve the architecture, design and coding of the product, and increasing performance testing.

CTFL-5.5.5 K2 Describe, using examples, how risk analysis and risk management may be used for test planning.

22 The answer is A.

- A** YES - the test manager and team should plan to carry out some data migration testing to make sure the process and data for prices being added to and updated on the website is correct, and also plan for reliability and performance testing to ensure the customer experiences a robust and timely response from the website. Risks ii, iii and vi are all PROJECT risks.

However, these are all important risks that affect the potential start date and progress for the project as a whole and for testing, so we would want in real life to discuss those with the rest of the project team, because they will need to be addressed by the team.

- B** NO - these risks are all PROJECT risks, and the risk based approach to test planning looks at PRODUCT risks to decide how much testing to plan to do. However, these are all important risks that affect the potential start date and progress for the project as a whole and for testing, so we would want in real life to discuss those with the rest of the project team, because they will need to be addressed by the team.
- C** NO – this is a product risk, but the list of identified risks also included product risks i and v, so these would also need to be assessed to decide what place they have in the testing.
- D** NO – this is a project risk, one of the important risks that affect the potential start date and progress for the project as a whole and for testing, so we would want in real life to discuss those with the rest of the project team, because they will need to be addressed by the team. To decide how much testing and what type of testing to do, in a risk based approach we look at the product risks.

CTFL-5.6.1 K1 Recognise the content of an incident report according to the 'Standard for Software Test Documentation' (IEEE Std. 829-1998).

23 The answer is A.

- A** YES – this is not part of the Test Incident Report as described in IEEE 829-1998, it is part of the Test Plan. The test items typically are whole items under test so this describes the decision making around release or not for a software system for example, so would include consideration of multiple test results and incident reports.
- B** NO – this is part of the Incident Description within a Test Incident report. The incident description is the third part of the Incident Report and has the structure:
 - Inputs
 - Expected results
 - Actual results
 - Anomalies
 - Date and time
 - Procedure step
 - Environment
 - Attempts to repeat
 - Testers
 - Observers.
- C** NO– this is part of the Incident Description within a Test Incident report. The incident description is the third part of the Incident Report and has the structure as given in B above.
- D** NO – this is the final section of the Incident Report. The structure of the full report is:
 - Test Incident Report Identifier
 - Summary
 - Incident Description
 - Impact.

24 The answer is A.

- A** YES – when the programmer looks at this report it will be useful for her to know what worked and what did not work. Was this a boundary problem? Knowing the effect of ordering fewer and more than 5 items will help the programmer establish this, and only one report is needed.
- B** NO – in this case, provided the tester has reproduced the problem, and it is easy to do that, it is likely that reproducing the problem is trivial. If it is difficult to reproduce, then detailed steps and environment description may be needed.
- C** NO – in this case, the calculations are not complex, and adding them in will not help the developer. In a test of system with complex calculations, then the worked example of the expected and actual results could be very useful if for example you are checking the algorithm used.
- D** NO – the test design method is not likely to be of interest at this point, and probably would be indicated by the description of the additional inputs mentioned in A. It is likely that BVA was used.

Practice Exam 6_2 Questions

- 1** Which of the following statements about tool support for testing is true?
- A** Static analysis tools can be used to aid test planning by providing information about risk areas in the code.
 - B** Dynamic analysis tools are mainly used during the live monitoring of production systems to identify memory leaks.
 - C** Test harnesses are used during system testing to drive keyword or data driven automation packs.
 - D** There is no tool support for software attributes such as usability, which has to be a manually driven process.
- K2 (1 mark)**
- 2** A head of IT wants to investigate whether tool support for testing will help the test team reduce test time and costs. Which of the following is NOT a reason for considering tool support?
- A** Tools could make increased static analysis possible by reducing the resources needed for code reviews.
 - B** Tools could improve the efficiency of test activities by automating the repetitive parts of test management.
 - C** Tools could increase reliability of testing by automating activities that humans find difficult.
 - D** Tools could reduce the elapsed time and budget for the current project, which is running behind schedule.
- K2 (1 mark)**
- 3** Which of the following are always potential benefits of tool support for testing?
- A** A tool can be used as a standalone support for testing.
 - B** A tool can reduce time scales and costs for a project.
 - C** A tool can replace test design activities.
 - D** A tool can provide objective measurement of coverage.
- K2 (1 mark)**
- 4** Which of the following best describes the implementation of a static analysis tool?
- A** Static analysis tools can provide a large quantity of data on potential code flaws very quickly so a full implementation provides that data quickly and early.
 - B** Static analysis tools require an input spreadsheet with keywords and data to drive the analysis, and this is best provided first.
 - C** Static analysis tools can provide a large quantity of data on potential code flaws, so can best be implemented in stages with filtering applied to results.
 - D** Static analysis tools provide information on physical data models, and implementation requires online checklists and scripting expertise.
- K1 (1 mark)**
- 5** Which of the following activities are best performed before a tool is acquired by an organisation, and which are best performed after the tool is acquired?
- i. Assessment of improvement opportunities
 - ii. Running a pilot project
 - iii. Evaluation of vendor and service support organisations
 - iv. Evaluation of how the tool fits with existing practices
 - v. Assessing whether the benefits will be achieved at reasonable cost
 - vi. Running a proof of concept

- vii. Evaluation of training needs
- viii. Deciding on naming conventions

- A** i, iii, vi, vii are best performed before a tool is acquired, and the rest after.
- B** ii, iv, v, and viii are best performed before a tool is acquired and the rest after.
- C** i, ii, iii are best performed before a tool is acquired, and the rest after.
- D** i, iii, v are best performed before a tool is acquired, and the rest after.

K1 (1 mark)

- 6** Which of these statements best differentiates between the goals of a proof of concept and a piloting phase?

- A** A pilot is done during evaluation to check the tool operates within the existing infrastructure and a proof of concept is done to introduce the tool once acquired to evaluate how the tool fits with existing processes.
- B** A proof of concept is done during evaluation to check the tool operates within the existing infrastructure and a pilot is done to introduce the tool once acquired to evaluate how the tool fits with existing processes.
- C** A proof of concept is done to identify changes to the infrastructure and a pilot is done to identify whether the tool performs effectively with the system under test.
- D** A proof of concept is done to learn more detail about the tool and decide on standardisation such as naming convention and a pilot is done to assess whether the benefits will be achieved at reasonable cost.

K1 (1 mark)

- 7** A test manager is asked to purchase and implement a test automation tool as quickly as possible across all her projects to speed up testing and reduce costs. Which of the following factors should she suggest to her manager and request time and budget for, to improve the success of the tool deployment?

- A** A big bang implementation, with training, coaching and mentoring provided.
- B** An incremental rollout with a pilot first, and supported by process improvement.
- C** A pilot and then a parallel rollout, with teams implementing common standards.
- D** An incremental rollout post pilot with each team defining their own standards.

K1 (1 mark)

Practice Exam 6_2 Answers

CTFL-6.1.1 K2 Classify different types of test tools according to their purpose and to the activities of the fundamental test process and the software life cycle.

1 The answer is A.

- A YES – where code already exists – for example if this is a maintenance or upgrade project – a static analysis tool can be used to detect areas of complexity in the code which may require additional testing, and also potential areas for tidying up the code, such as modules that cannot be reached.
- B NO – dynamic analysis tools are used to identify memory leaks and similar phenomena, and can be used for live monitoring, but have a significant role to play in component and component integration testing. We would expect that memory leaks are tested for and fixed as early as possible.
- C NO – test harnesses are reusable testing libraries used in building testing tools. They are a type of framework. Another type of framework is the test automation framework, and this is where data driven and keyword driven approaches are useful.
- D NO - tool support for usability is available, although this is quite specialist and can include the use of EEG monitoring for brain activity monitoring and eye tracking software, as well as recording & playback of hand/mouse/keyboard movements, recording of commentary, and multiple views on a single screen of user activity, to be viewed by an observer in another room.

CTFL-6.1.2 K2 Explain the term test tool and the purpose of tool support for testing.

2 The answer is D.

- A NO – this is a benefit of tools, static analysis done manually by code review is very time consuming, and especially the syntax analysis of source code (which used to be a manual process in the early days) was quickly integrated into pre-compiler tools to allow both a faster and more accurate syntax analysis pre-compile. Other forms of static analysis are also time-consuming, or even near-impossible for humans to complete accurately. However, a code review by humans is still useful for both finding and debugging problems.
- B NO – automating test management activities is highly beneficial and can be a good starting point, rather than automating test execution. The more mundane and repetitive parts of test management for example compiling reports from multiple data sets is done more quickly and more accurately by a tool once set up.
- C NO – many parts of testing are repetitive and humans quickly start to make mistakes as they lose concentration or to inadvertently change the steps they are taking. However, it is important to realise that the most valuable things that a human brings to testing – intelligence, ability to question and analyse, problem solving – are also hard to do, but rewarding to do, and are much harder to emulate with a tool than the repetitive areas.
- D YES – this answer is a wrong expectation of the benefits of tools, although a common misconception among those procuring tools. The process of evaluating and piloting the tools will cost time and money, and even if this is not directly paid for through the project's budgets, it is highly unlikely that the benefits would apply to the initial project.

3 The answer is D.

- A NO – although some tools are standalone, the greatest benefit comes when a tool set can be integrated. This can be across the development life cycle activities – for example requirement tools used by Bas sharing change data with the test repository automatically – or within testing for example test execution results populating a test management reporting database.
- B NO – cost and time scale reduction is not the only – and not necessarily the best – reason for implementing a tool. For example the key benefit may be to increase quality of the product and reliability of the testing, in which case we may budget for additional activities and costs in our projects. For some specialist tools we may require specialist skills which cost more to both run and analyse results (performance, security, usability tools). We may need to spend more time and effort on better test design. And finally, the tool implementation project will initially cost before the investment pays back.
- C NO – although there are some test design tools, the activities of an intelligent human brain to pose questions, identify risk areas, devise interesting tests building on those auto-generated is highly important.
- D YES – coverage of the code by test execution is always more accurate when done by a tool. Different tools and different coverage measures may produce different results on the same piece of code, because they are implemented to measure in different ways, but – having chosen your tool, as you measure different piece of code with the same tool and coverage measurement, the results will be consistent.

4 The answer is C.

- A NO – if all the reports from the static analysis tool are received at once it is very likely that the team will be overwhelmed and discouraged, and therefore rejects all the reports and the tool.
- B NO – static analysis tools do not require an input spreadsheet – the use of keywords and data is required to drive test automation frameworks.
- C YES – especially if unleashed on old and ill formed code, a static analysis tool is highly likely to produce a very large of warnings, as well as a few failures in the code. When developers compile their code, they often switch off/filter the warnings they know to be unimportant, so that they can concentrate on the failures and the important warnings. Once those are resolved, they may go back to less important warnings and resolve those. In the same way, the filters on a static analysis tool are a useful way to allow a team to concentrate on the most important reports from the static analysis tool first.
- D NO – static analysis tools do not provide information on physical data models (pre code artefact). This is likely to be provided by a modelling tool. Static analysis tools do not require online checklists (the analysis rules are likely to be incorporated in the tool and its settings) but review tools may support online checklists for people to use during the review process. Finally, scripting

expertise is more likely to be needed for test automation rather than for static analysis tools.

CTFL-6.3.1 K1 **State the main principles of introducing a tool into an organisation.**

5 The answer is A.

- A YES – a pilot project is the first activity after acquiring a tool, and that will include evaluating the tool in actual use against existing practices, assessing whether the actual cost benefits meet the cost benefits assessed for the business case, and agreeing naming conventions. The remaining actions in the list are best done before the tool is acquired.
- B NO - a pilot project is the first activity after acquiring a tool, and that will include evaluating the tool in actual use against existing practices, assessing whether the actual cost benefits meet the cost benefits assessed for the business case, and agreeing naming conventions. The remaining actions in the list are best done before tool is acquired.
- C NO – running the pilot project has to be done after the tool is acquired. The proof of concept will be done before the tool is acquired to check technical integration of the tool will work. Evaluating training needs is also done before the tool is acquired – both for assessing potential costs and for assessing comparative training needs in a short list of tools.
- D NO – assessing the actual cost benefit can only be done once the tool has been acquired. A cost benefit estimate would form part of the business case before the tool is acquired. One reason for the pilot is to assess how close the actual cost benefit is the estimated one. The proof of concept will be done before the tool is acquired to check technical integration of the tool will work. Evaluating training needs is also done before the tool is acquired – both for assessing potential costs and for assessing comparative training needs in a short list of tools.

CTFL-6.3.2 K1 **State the goals of a proof-of-concept for tool evaluation and a piloting phase for tool implementation.**

6 The answer is B.

- A NO – the proof of concept is done before a tool is acquired to make sure it fits within the architecture and to identify changes to the architecture. The pilot is done once the tool is acquired, and is the first step in rollout. It is done to identify changes to processes, among other things.
- B YES – the proof of concept is done before a tool is acquired to make sure it fits within the architecture and to identify changes to the architecture. The pilot is done once the tool is acquired, and is the first step in rollout. It is done to identify changes to processes, among other things.
- C NO – A proof of concept does identify changes to the infrastructure, but it will also include identifying whether the tool performs effectively with the tool. It would be too late to do this in the pilot.
- D NO – A pilot does help assess whether the actual cost benefits and it also is used to identify naming conventions and find out more about the tool. The proof of concept is too early to invest a lot of time in a group of people learning the tool, and until it is used in earnest what is required in standardisation and naming conventions may not be apparent.

7 The answer is B.

- A NO – providing training, coaching and mentoring is good practice, but a big bang approach without a pilot is likely to fail because the pilot identifies information that allows tailoring of the plan based on actual tool usage. Also, it is likely to cause a heavy strain on the organisations, with all teams requiring training, coaching and mentoring at the same time, and the pilot project is unlikely to allow time for that. The end result is likely to be either that all testing progress is slowed or even halted to allow the automation implementation, or that the automation project is abandoned.
- B YES – although taking an incremental approach looks slower, the end result will be more robust and the whole process is more likely to be successful. Supporting the tool implementation with process improvement means we are more likely to be doing the right things as effectively and as efficiently as possible, rather than automating undesirable practices.
- C NO – whilst a pilot project is good practice, as is having common standards across all the teams, choosing a parallel rollout to all teams is likely to cause a heavy strain on the organisations, with all teams requiring training, coaching and mentoring at the same time, and the pilot project is unlikely to allow time for that. The end result is likely to be either that all testing progress is slowed or even halted to allow the automation implementation, or that the automation project is abandoned.
- D NO – whilst using a pilot project and an incremental approach is good practice suggesting that each team can define its own naming conventions and other standards would be a mistake. Although allowing teams discretion to choose their own standards will seem to save time by reducing inter-team discussion, it would have the following disadvantages long term:
- Team members will be less transferable between teams as they will learn different conventions and standards.
 - Automation packs for common components of the systems under test would not be transferable between teams, potentially meaning that teams would duplicate work.
 - Some teams may end up working with suboptimal conventions, standards and processes.
 - Measurement and comparison across teams will be less likely to be possible in a meaningful way.

Mock Exam

SHORT SAMPLE PAPER FOR DAY 1

Note: this shortened paper has been created using questions from actual sample papers provided by BCS

Time allowed: 38 minutes

Record your surname/last/family name and initials on the Answer Sheet, writing in block capitals at the top and marking the relevant letter in each column.

Attempt all 25 multiple-choice questions – 1 mark awarded to each question.

A number of possible answers are given for each question, indicated by either **A. B. C.** or **D.** Only **one** of these answers and no more than one answer is correct in each multiple-choice question.

Your answers should be indicated on the Answer Sheet by making a **solid pencil mark** inside the box representing your chosen answer. These can be found to the right of the question number on your Answer Sheet.

If you make a mistake, erase your first mark, and put a mark in your new chosen answer column for that question.

Mark **only one answer** to each question.

Pass mark is 17/25

1 Which of the following is a characteristic of good testing in any life cycle model?

- A Analysis and design of tests begins as soon as development is complete.
 - B Some, but not all, development activities have corresponding test activities.
 - C Each test level has test objectives specific to that level.
 - D All document reviews involve the development team.
-

2 Which of the following defines the sequence of actions for the execution of a test?

- A Test case specification.
 - B Test design specification.
 - C Test procedure specification.
 - D Test plan.
-

3 What is the objective of debugging?

- i. To identify the cause of a defect.
- ii. To fix a defect.
- iii. To show values.
- iv. To increase the range of testing.

- A ii, iv.
 - B ii, iii, iv.
 - C i, ii.
 - D i, iii.
-

4 Which of the following statements is independent of the development life cycle model being used?

- A Testers should review all development documents.
 - B The majority of development activities have an associated testing activity.
 - C Design of tests should begin after the corresponding development activity has been completed.
 - D Each test level has objectives specific to that level.
-

5 Which of the following are valid objectives for testing?

- i. To find defects.
- ii. To gain confidence in the level of quality.
- iii. To identify the cause of defects.
- iv. To prevent defects.

- A ii, iii and iv.
 - B i, ii and iv.
 - C i,iii and iv.
 - D i,ii, and iii.
-

6 When a defect in code is executed, which of the following will be the result?

- A** Fault.
 - B** Error.
 - C** Failure.
 - D** Mistake.
-

7 Why are static testing and dynamic testing described as complementary?

- A** Because they have different aims and differ in the types of defect they find.
 - B** Because they share the aim of identifying defects and find the same types of defect.
 - C** Because they have different aims but find the same types of defect.
 - D** Because they share the aim of identifying defects but differ in the types of defect they find.
-

8 In which activity of the fundamental test process is the test basis reviewed?

- A** Test closure activities.
 - B** Test planning and control.
 - C** Evaluating exit criteria and reporting.
 - D** Test analysis and design.
-

9 What is the difference between a data-driven approach to test execution and a keyword-driven approach?

- A** A data-driven approach uses action words with data; a keyword-driven approach separates out the test inputs into a spreadsheet.
 - B** A data-driven approach uses a generic script with separate sets of input data; a keyword-driven approach uses keywords describing actions with test data.
 - C** A data-driven approach uses a spreadsheet to associate specific actions with action words; a keyword-driven approach uses a spreadsheet to identify sets of data.
 - D** A data-driven approach uses generic actions with one set of specific data; a keyword-driven approach uses a specific set of actions with multiple data sets.
-

10 Which of the following is associated with the Test Planning and Control activity of the Fundamental Test Process?

- A** Defining tangible test conditions and test cases.
 - B** Setting up the test environment.
 - C** Defining the objectives of testing.
 - D** Specifying test procedures or scripts.
-

11 What is the main benefit of independent testing?

- A** It helps to avoid author bias – testing from just their point of view.
 - B** It ensures that the maximum number of defects is found.
 - C** It provides an opportunity to check that standards have been adhered to.
 - D** It helps to educate developers from other projects if they are exposed to other developers' work.
-

12 What is the key difference between alpha testing and beta testing?

- A** Alpha testing is performed at the developer's site; beta testing is performed at the customer's site.
 - B** Alpha testing is performed by developers; beta testing is performed by customers.
 - C** Alpha testing is performed by customers; beta testing is performed by developers.
 - D** Alpha testing is performed at the customer's site; beta testing is performed at the developer's site.
-

13 Which activity in the fundamental test process includes evaluation of the testability of the requirements and system?

- A** Test analysis and design.
 - B** Test implementation and execution.
 - C** Test closure.
 - D** Test planning and control.
-

14 Which of the following statements is true with respect to component testing?

- A** Component tests cannot be prepared before the component is coded.
 - B** Component testing does not normally require access to the code being tested.
 - C** Component testing may include testing of functionality and specific non-functional characteristics.
 - D** Component testing is normally subject to formal incident reporting.
-

15 Which of the following statements correctly characterise regression testing?

- a. Regression testing is only performed at the system testing level.
 - b. Regression testing applies to functional, non-functional and structural testing.
 - c. Regression testing should be automated.
 - d. Regression test suites often evolve slowly over time.
 - e. Regression testing helps to ensure that changes to software have been correctly implemented.
- A** c and d.
 - B** a and c.
 - C** b and e.
 - D** b and d.
-

Mock Exam Day 1

16 Which of the following defines the scope of maintenance testing?

- A** The time since the last change was made to the system.
 - B** The coverage of the current regression pack.
 - C** The size and risk of any change(s) to the system.
 - D** Defects found at the last regression test run.
-

17 During which fundamental test process activity do we determine if MORE tests are needed?

- A** Evaluating test exit criteria.
 - B** Test analysis and design.
 - C** Test implementation and execution.
 - D** Test planning and control.
-

18 If system testing has identified several defects in one area of the system and time is limited, where should you concentrate effort to find new defects?

- A** In areas as remote as possible from where defects have been found.
 - B** In the area where defects have already been found.
 - C** In areas of the application selected at random.
 - D** In well-tested areas where no defects have been found.
-

19 Which of the following is a MAJOR activity of test planning?

- A** Evaluating exit criteria and reporting.
 - B** Determining the test approach.
 - C** Preparing test specifications.
 - D** Measuring and analyzing results.
-

20 Which of the following statements about regression testing is FALSE?

- A** Regression testing should be performed whenever software is changed.
 - B** Regression tests should be reviewed to ensure that they are still relevant to the business.
 - C** Automated regression testing is always more efficient than manual regression testing.
 - D** Regression tests can be run many times.
-

21 Which of the following is a benefit of independent testing?

- A** Code cannot be released into production until independent testing is complete.
 - B** Testing is isolated from development.
 - C** Independent testers see other and different defects, and are unbiased.
 - D** Developers do not have to take as much responsibility for quality.
-

22 When should maintenance testing be carried out?

- A** When any software is modified to remove defects.
 - B** When deployed software or its environment changes.
 - C** When deployed software has been in service for an extended period.
 - D** When changes to requirements occur late in a development project.
-

23 A project is using an iterative-incremental development model. Which of the following characterises an iterative development model:

- a. Regression testing is increasingly important on all iterations after the first
- b. The resulting system produced may be tested at several levels as part of the development
- c. Verification and validation are not carried out on increments
- d. An increment, added to others developed previously, forms a growing partial system

- A** a, b and c.
 - B** a, c and d.
 - C** a, b and d.
 - D** b, c and d.
-

24 Which of the following kinds of tool can be used to find defects that are evident only when software is executing?

- A** Coverage measurement tool.
 - B** Static analysis tool.
 - C** Dynamic analysis tool.
 - D** Modelling tool.
-

25 Why does pure “capture-replay using a linear script approach in automation not scale to large numbers of scripts?

- A** Because each script captures specific data and is not stable when unexpected events occur.
 - B** Because each script captures specific data and is stable when unexpected events occur.
 - C** Because each script captures generic data and is not stable when unexpected events occur.
 - D** Because each script captures generic data and is stable when unexpected events occur.
-

End of Paper

Mock Exam Answers

Commentary on the Questions - Sample Paper Day 1

Question 1

The correct answer is C.

Be careful of wording of the answers. The reason why C is the correct answer is that each test level has objectives to achieve and these may be different to other test levels. For example, acceptance testing has the objective of gaining confidence; system testing has a different objective, i.e. gain confidence in addition to finding defects.

Question 2

The correct answer is C.

This is testing the delegate's knowledge on the testing process. The document that has a sequence of action for the execution is a test procedure specification. Test will describe the pre-condition, inputs and expected results (with post-conditions) for one test and is the stage after creating test conditions. Test design specification is another term for test case specification. In some organisations the sequence of actions for executing a test is called a test plan, however you will be tested on terminology according to ISTQB.

Question 3

The correct answer is C.

The first two options were described in the slides regarding testing and debugging, option iii. Doesn't make any sense, to show values and iv) debugging removes the defect from the code not ensure more coverage through testing.

Question 4

The correct answer is D.

Answer A is not correct; in some cases it may be valuable for testers to review some development documents but it is never the case that testers should review all development documents; this is also a statement that reflects a sequential life cycle rather than an iterative life cycle in which development documents would play a less significant role. Answer B is incorrect because **all** development activities should have an associated testing activity. Answer C reflects a reactive approach and in some life cycles (the V-model, for example) we would expect design of tests to begin before the corresponding development activity has been completed. Answer D is correct and reflects the wording of the syllabus.

The syllabus reference is section 2.1.

Grove comment: we believe this is a bad question and as such if you get this question wrong, please do not worry about it.

Question 5

The correct answer is B

All of the statements are valid objectives except iii) to identify the cause of a defect. This is done by developers and performed at any stage of the development and testing when a failure has occurred. It is part of debugging not testing.

Question 6

The correct answer is C.

An error is a human mistake that leads to a defect, so D is a synonym for error (answer B) and neither is correct. A fault (answer A) is another name for a defect, used in the question, so that is incorrect. A failure is a manifestation of a defect when code is executed, so C is the correct answer.

The syllabus reference is section 1.1.

Question 7

The correct answer is D. This is a K2 question.

This question requires you to be able to differentiate between static and dynamic testing in respect of their aims and the type of defects they find. They do share the same aim, that of identifying defects, but they find different types of defect. You could use reason to get to this, since there would be little purpose in having both static and dynamic testing to find the same type of defects, and you might also recognise that one purpose of all testing is to identify defects.

The syllabus reference is section 3.1.

Question 8

The correct answer is D. This is a K1 question.

This question requires recall of the stages of the fundamental test process and the main activities at each stage. The test basis is the starting point for testing, so it needs to be reviewed before test conditions are identified, which is part of test analysis and design. Test closure and evaluating exit criteria and reporting can both be eliminated if the nature of the test basis is recognised. Test planning and control is a stage that begins even earlier than test analysis and design but is a different kind of activity and the name should help to eliminate that option.

The syllabus reference is section 1.4.

Question 9

The correct answer is B. This is a K2 question.

This question requires knowledge of the nature of both data-driven and keyword-driven testing. Answer B identifies the correct characteristics. Data-driven testing associates multiple sets of data with a set of actions (usually with a spreadsheet). Looking only at the data-driven part of each answer, answers A and B are feasible but C and D are incorrect. Keyword-driven testing associates sets of actions with keywords which eliminates answer A. Answer B combines a correct definition of data-driven testing with a correct definition of keyword-driven testing.

The syllabus reference is section 6.2.

Question 10

The correct answer is C. This is a K1 question

This question requires recall of the stages of the fundamental test process and the main activities at each stage, relating to the planning activities. In this question answers A and D are clearly test analytical in nature, while option B would come later in the process. Answer C is the only response that is about planning.

The syllabus reference is section 1.4.

Question 11

The correct answer is A. This is a K1 question.

This is a recall question and answer A is a statement based on the wording in the syllabus. Answer B is incorrect because independence cannot guarantee the number of defects found; it is a distracter because we would hope that independence would increase the number of defects found but there is no guarantee that this is the case; the defect count would depend on the quality of testing and independent testing is not automatically better than any other kind, only different. Answers C and D might both be considered benefits but neither is a 'main' benefit because neither makes any immediate contribution to finding defects. Standards adherence and education are both attractive side effects if they occur but neither could justify the cost of independent testing.

The syllabus reference is section 1.5.

Question 12

The correct answer is A. This is a K2 question.

This question requires that two aspects of alpha and beta testing are understood: where it happens and who does it. Answers B and C are both incorrect because both alpha and beta testing are performed by customers. Alpha testing is performed at the developer's site, so A is correct and D is incorrect. Hence answer A is the correct answer.

The syllabus reference is section 2.2.

Question 13

The correct answer is A This is a K1 question.

This is a question requiring recall of the fundamental test process. Test analysis and design is where testability of requirements is checked. You could reason this on the basis that testability of requirements should be checked as early as possible, eliminating answers B and C. We then have a choice between a planning activity and an analysis activity, and checking testability is an analysis activity, so A is correct rather than D.

The syllabus reference is section 1.4.2.

Question 14

The correct answer is C. This is a K1 question.

Answer C can be arrived at by recall. Alternatively, we could eliminate answer A because some component testing can be prepared from the component specification; answer B is incorrect because component testing is typically based on the code; answer D is incorrect because formal incident reporting is sometimes not initiated until after component testing.

The main syllabus reference is section 2.2.

Question 15

The correct answer is D. This is a K2 question.

This question can be approached by elimination. The key fact is that regression testing is done to ensure that unintended changes have not been introduced. Answer a is incorrect because regression testing is needed whenever code is changed; answer b is correct because regression suites should cover all aspects of a system after change; answer c is incorrect because, although automation is an attractive option, it is neither required nor necessary; answer d is correct because the regression suite must match the changing functionality of a system as it evolves; answer e is incorrect because the statement describes re-testing rather than regression testing. Thus answers b and d are correct and this corresponds to answer D.

The syllabus reference is section 2.3.

Question 16

The correct answer is C. This is a K1 question.

This is a recall question. There is no simple way to eliminate incorrect answers but an awareness that scope determines what should be tested would help to eliminate answers B and D, both of which are about the effectiveness of regression testing as a tool of maintenance testing. Answer A is irrelevant; time does not trigger maintenance testing if nothing has changed.

The syllabus reference is section 2.4.

Question 17

The correct answer is A. This is a K1 question.

This question relates to the point in the fundamental test process where the requirement for MORE tests is identified. The emphasis on MORE tests steers us away from the point where the number of test required is initially determined, which would be test planning (answer D). Test analysis and design is where test cases are generated and test implementation and execution is where tests are executed, so both of these phases would help to generate the data needed to decide whether extra tests are needed, but it is when exit criteria are checked that the decision to do more testing would be taken (if exit criteria are not met). So, the correct answer is A.

The syllabus reference is section 1.4.4.

Question 18

The correct answer is B. This is a K1 question.

This question relates to one of the key principles of testing, namely that defects cluster. This leads directly to answer B. There is no way to eliminate alternatives if this principle is not known; this K1 question relies on recall of the principle.

The syllabus reference is section 1.3.

Question 19

The correct answer is B. This is a K1 question.

This is another question about the fundamental test process, this time concerning test planning. We can eliminate answers A and D, since both need test results, so the choice is between B and C. Answer C is about test analysis and design, so answer B is correct. Alternatively, having a good grasp of the nature of test planning would guide you to answer B as the one most closely related to the most important purpose of test planning.

The syllabus reference is section 5.2.2.

Question 20

The correct answer is C. This is a K1 question.

This is a recall question about the nature of regression testing. It is an example of an unusual style of question – the negative question. It could have been rephrased as ‘which 3 of the following statements are true?’ but the negative question is, in this case, clearer. If we know that regression testing is about testing software after a change to determine whether any unwanted affects have been introduced in areas that were not changed we can identify answer A as true. Answer B sounds as if it should be true; there would be little point in running a test that had no value. Answer D is also true because regression tests are required whenever software changes. This leaves C. In many cases automated regression testing will be more efficient than manual regression but this cannot be guaranteed – it is not necessarily always true. Thus answer C is an incorrect statement and is therefore the correct answer to the question.

The syllabus reference is section 2.3.4.

Question 21

The correct answer is C This is a K1 question.

This is a recall question about independent testing. Answer C is taken directly from the syllabus; recognising this provides the correct answer directly. Alternatively, we can reason that answer A is incorrect because it would simply that independent testing is always mandatory and this is not true. Answer B suggests an approach that contradicts other parts of the syllabus, where co-operation is promoted. Answer D would reduce the developers’ responsibility for quality, which also contradicts the idea that everyone who creates a work product must take responsibility for its quality.

The syllabus reference is section 5.1.1.

Question 22

The correct answer is B. This is a K1 question.

Maintenance testing is any testing carried out on software after its initial release for live use. Answer A is not completely correct because it specifies ‘to remove defects’ which is not a reason for maintenance because it could happen during the development project. Answer B is the correct description of when maintenance testing is necessary. Answer C is incorrect because it specifies a time period and maintenance can begin immediately on deployment (if changes to the software are approved). Answer D is incorrect because it is related to the development project.

The syllabus reference is section 2.4.

Question 23

The correct answer is C. This is a K2 question.

The wording of three of the options is taken directly from the syllabus, and it is a matter of establishing which of the options is incorrect. The syllabus does not say that verification and validation are not carried out, just that they may be carried out. Answer C is the only answer that excludes the wrong choice (option c). As a check, the other items (regression testing is increasingly important, the resulting system produced may be tested as part of development, and an increment added to what was present hitherto forms a growing partial system) are specifically mentioned in the syllabus.

The syllabus reference is section 2.1.

Question 24

The correct answer is C. This is a K1 question.

This question requires knowledge of the categories of tools and their basic characteristics. If software must be executing as defined in the question then static analysis tools are inappropriate, since 'static' implies 'not executing'. Similarly 'modelling' is an activity associated with analysis and design and not with test execution. Coverage measurement implies that tests have been executed, but the measurement of coverage is an activity that determines the actual coverage achieved during test execution. Coverage measurement is used to determine the completeness of testing, not to find defects. Thus dynamic analysis ('dynamic' implies the execution of the code under test) is the correct answer (answer C).

The syllabus reference is section 6.1.

Question 25

The correct answer is A. This is a K2 question.

This question requires an understanding of the nature of capture-replay as a means of creating test scripts. Recording the actions of a manual tester clearly captures specific rather than generic data, so answers C and D are incorrect. Will the captured data and actions be stable or unstable? If something unexpected occurs the test will not have been set up to cope with that event. Some events of this kind may be relatively benign, but some may cause the test to fail and the test script to terminate. This kind of instability is always a risk for test scripts that have not been designed to react appropriately, so instability will always be a potential feature of scripts captured by recording the actions of a manual tester. Thus the correct answer is A.

The syllabus reference is section 6.2.

End of Commentary

SAMPLE PAPER FOR DAY 2

Note: this paper has been created using questions from actual sample papers provided by BCS, the Chartered Institute for IT, with additional questions provided by Grove Consultants

Time allowed: 1 hour

Record your surname/last/family name and initials on the Answer Sheet, writing in block capitals at the top and marking the relevant letter in each column.

Attempt all 40 multiple-choice questions – 1 mark awarded to each question.

A number of possible answers are given for each question, indicated by either **A. B. C.** or **D.** Only **one** of these answers and no more than one answer is correct in each multiple-choice question.

Your answers should be indicated on the Answer Sheet by making a **solid pencil mark** inside the box representing your chosen answer. These can be found to the right of the question number on your Answer Sheet.

If you make a mistake, erase your first mark, and put a mark in your new chosen answer column for that question.

Mark **only one answer** to each question.

Pass mark is 26/40

1 Given the following State Table:

	A	B	C	D	E	F
SS	S1					
S1		S2				
S2			S3		S1	
S3				ES		S3
ES						

Which of the following represents an INVALID state transition?

- A** E from State S2.
 - B** F from State S3.
 - C** B from State S1.
 - D** E from State S3.
-

2 Consider the following techniques. Which are static and which are dynamic techniques?

- i. Equivalence Partitioning.
- ii. Use Case Testing.
- iii. Data Flow Analysis.
- iv. Exploratory Testing.
- v. Decision Testing.
- vi. Inspections.

- A** iii and vi are static, i, ii, iv and v are dynamic.
 - B** vi is static, i-v are dynamic.
 - C** i, ii, iii and iv are static, v and vi are dynamic.
 - D** ii, iii and vi are static, i, iv and v are dynamic.
-

3 Which of the following describes specification-based test design techniques?

- A** Test cases are derived from an analysis of both the test basis documentation and the structure of the component or system.
 - B** Test cases are derived mainly from the tester's experience.
 - C** Test cases are derived from an analysis of the structure of a component or system.
 - D** Test cases are derived from an analysis of the test basis documentation of a component or system.
-

4 Which of the following statements is MOST OFTEN true?

- A** Component testing is an important part of user acceptance testing.
 - B** Component testing searches for defects in programs that are separately testable.
 - C** Source-code inspections are often used in component testing.
 - D** Component testing aims to expose problems in the interactions between software and hardware components.
-

5 How can early testing **best** help to prevent defects?

- A The intellectual exercise of designing tests can identify defects in design documentation that could have given rise to defects in code.
 - B Identifying defects in design documents shows developers that the test team is technically competent and this makes them more careful in their work.
 - C Testing of requirements ensures that they are complete, consistent and defect-free as a basis for development.
 - D Early test design familiarises testers with the design and so enables them to test code more effectively.
-

6 Given the following sample of pseudo code:

```
Input Number_of_Coins
Total = 0
While Number_of_Coins > 0
    Input Value_of_Coin
    Total = Total + Value_of_Coin
    Number_of_Coins = Number_of_Coins - 1
End Loop
Print "Your coins are worth" & Total
```

What is the minimum number of test cases required to guarantee 100% decision coverage?

- A 3.
 - B 1.
 - C 4.
 - D 2.
-

7 What is the MAIN purpose of impact analysis for testers?

- A To determine how much the planned changes will affect users.
 - B To determine how the existing system may be affected by changes.
 - C To determine what proportion of the changes need to be tested.
 - D To determine the programming effort needed to make the changes.
-

8 Which of the following requirements would be tested by a functional system test?

- A The system must allow 12,000 new customers per year.
 - B The system must allow a user to amend the address of a customer.
 - C The system must perform adequately for up to 30 users.
 - D The system must be able to perform its functions for an average of 23 hours 50 minutes per day.
-

9 Given the following decision table:

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
UK resident?	False	True	True	True
Age between 18 - 55?	Don't care	False	True	True
Smoker?	Don't care	Don't care	False	True
Actions				
Insure client?	False	False	True	True
Offer 10% discount?	False	False	True	False

What is the expected result for each of the following test cases?

- A. TC1: Fred is a 32 year old smoker resident in London
- B. TC3: Jean-Michel is a 65 year non-smoker resident in Paris

- A A – Don't insure, B – Don't insure.
 - B A – Insure, no discount, B – Don't insure.
 - C A – Insure, no discount, B – Insure with 10% discount.
 - D A – Insure, 10% discount, B – Insure, no discount.
-

10 In which of the following orders would the phases of a formal review usually occur?

- A Planning, kick off, preparation, meeting, rework, follow up.
 - B Kick off, planning, preparation, meeting, rework, follow up.
 - C Preparation, planning, kick off, meeting, rework, follow up.
 - D Planning, preparation, kick off, meeting, rework, follow up.
-

11 The process of designing test cases consists of the following activities:

- i. Elaborate and describe test cases in detail by using test design techniques.
- ii. Specify the order of test case execution.
- iii. Analyse requirements and specifications to determine test conditions.
- iv. Specify expected results.

According to the process of identifying and designing tests, what is the correct order of these activities?

- A iii, iv, i, ii.
 - B iii, ii, i, iv.
 - C iii, i, iv, ii.
 - D ii, iii, i, iv.
-

12 Given the following decision table:

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Frequent Flyer Member	Yes	Yes	No	No
Class	Business	Economy	Business	Economy
Actions				
Offer upgrade to First	Yes	No	No	No
Offer upgrade to Business	N/A	Yes	N/A	No

What is the expected result for each of the following test cases?

- A. Frequent flyer member, travelling in Business class
- B. Non-member, travelling in Economy class

- A** A – Don't offer any upgrade, B – Don't offer any upgrade.
 - B** A – Offer upgrade to First, B – Don't offer any upgrade.
 - C** A – Offer upgrade to First, B – Offer upgrade to Business class.
 - D** A – Don't offer any upgrade, B – Offer upgrade to Business class.
-

13 Which of the following will NOT be detected by static analysis?

- A** Unreachable (dead) code.
 - B** Variables that are never used.
 - C** Programming standards violations.
 - D** Errors in requirements.activities and tasks?
-

14 Which of the following tasks would typically be performed by a test design tool?

- i. Consistency checking.
- ii. Generate test data.
- iii. Generate test cases from software specification.
- iv. Generate test cases based on an analysis of the code.

- A** i and iv.
 - B** ii and iii.
 - C** iv only.
 - D** iii and iv.
-

- 15** Considering the following pseudo-code, calculate the MINIMUM number of test cases for statement coverage, and the MINIMUM number of test cases for decision coverage respectively.

```
READ A
READ B
READ C
IF C>A THEN
  IF C>B THEN
    PRINT "C must be smaller than at least one number"
  ELSE
    PRINT "Proceed to next stage"
  ENDIF
ELSE
  PRINT "B can be smaller than C"
ENDIF
```

- A** 2, 4.
 - B** 3, 3.
 - C** 2, 3.
 - D** 3, 2.
-

- 16** Which of the following statements highlight key differences between walkthroughs and technical reviews?

- a. Technical reviews may be run as a peer review;
- b. Walkthroughs may vary in practice from informal to very formal;
- c. Walkthroughs are led by the author;
- d. Technical reviews are used to evaluate alternatives and find defects;
- e. Walkthroughs may involve pre-meeting preparation.

- A** a and d.
 - B** b and e.
 - C** c and d.
 - D** b and c.
-

- 17** Which of the following is typical of the types of defect discovered by a static analysis tool?

- A** Requirements inconsistencies.
 - B** Inconsistent interfaces between modules.
 - C** Violations of the standards for specifications.
 - D** Incorrect input data.
-

18 In a system designed to work out the tax to be paid:

An employee has £4000 of salary tax free.
The next £1500 is taxed at 10%.
The next £28000 after that is taxed at 22%.
Any further amount is taxed at 40%.

To the nearest whole pound, which of these is a valid Boundary Value Analysis test case?

- A** £32001.
 - B** £28000.
 - C** £1500.
 - D** £33501.
-

19 Which of the following statements about the test development process is correct?

- A** Test cases are identified during test analysis; test conditions are specified in test design; test data is defined during test implementation.
 - B** Test conditions are identified during test analysis; test cases are specified in test design; test scripts are created during test implementation.
 - C** Test conditions are identified during test analysis; test scripts are created during test design; test data is specified during test implementation.
 - D** Test cases are identified during test analysis; test data is created during test design; test scripts are created during test implementation.
-

20 In a system designed to work out the tax to be paid:

An employee has £4000 of salary tax free.
The next £1500 is taxed at 10%.
The next £28000 after that is taxed at 22%.
Any further amount is taxed at 40%.

To the nearest whole pound, which of these groups of numbers fall into three DIFFERENT equivalence classes?

- A** £28000; £28001; £32001.
 - B** £4000; £5000; £5500.
 - C** £4000; £4200; £5600.
 - D** £32001; £34000; £36500.
-

21 Given the following State Table:

	E1	E2	E3	E4	E5	E6
SS	S1		S2			
S1		S2		S4		
S2			S3		S1	
S3				ES		S3
ES			SS			

Which of the following represents an **invalid** state transition?

- A Event E4 from state S3.
- B Event E2 from state S1.
- C Event E4 from state S2.
- D Event E3 from state SS.

22 Given the following decision table:

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Foundation exam pass	False	True	True	True
Intermediate exam pass	False	False	True	True
Practitioner exam pass	False	False	False	True
Actions				
Enter Practitioner exam.	False	False	True	False
Award Practitioner Cert.	False	False	False	True

What is the expected result for each of the following test cases?

- A: Jill holds the Foundation certificate but has not yet passed her Intermediate exam.
- B: Jack holds the Intermediate certificate but has not yet taken his Practitioner exam.

- A A – Do not award a Practitioner certificate to Jill or enter her for the Practitioner exam; B – Enter Jack for the Practitioner exam.
- B A – Enter Jill for the Practitioner exam; B – Enter Jack for the Practitioner exam.
- C A – Enter Jill for the Practitioner exam; B – Award a Practitioner Certificate to Jack.
- D A – Do not award a Practitioner certificate to Jill or enter her for the Practitioner exam; B – Do not award a Practitioner certificate to Jack or enter him for the Practitioner exam.

- 23** A candidate sits a multiple choice exam with 65 questions with variable points value totalling 100 points.

To pass, the candidate must achieve a score of at least 60 points.

To gain a distinction, the candidate must achieve a score of at least 80 points.

Which **one** of these groups of exam scores would fall into three **different** equivalence classes?

- A** 60, 79, 81.
 - B** 58, 74, 80.
 - C** 50, 60, 79.
 - D** 59, 80, 91.
-

- 24** Given the following sample of pseudo code:

```
01 Read the number of languages spoken
02 Read the number of countries visited
03 If number of languages spoken > 1 and number of countries visited > 0 then
04     Input "Do you want to travel more? (Yes or No)"
05     If "Yes"
06         Issue passport
07     End if
08 End If
```

Which of the following test cases will ensure that statement "06" is executed?

- A** Languages spoken = 1, countries visited = 0, travel more = "Yes".
 - B** Languages spoken = 1, countries visited = 1, travel more = "No".
 - C** Languages spoken = 2, countries visited = 2, travel more = "No".
 - D** Languages spoken = 3, countries visited = 2, travel more = "Yes".
-

- 25** The following statements are used to describe the basis for creating test cases using either black or white box techniques:

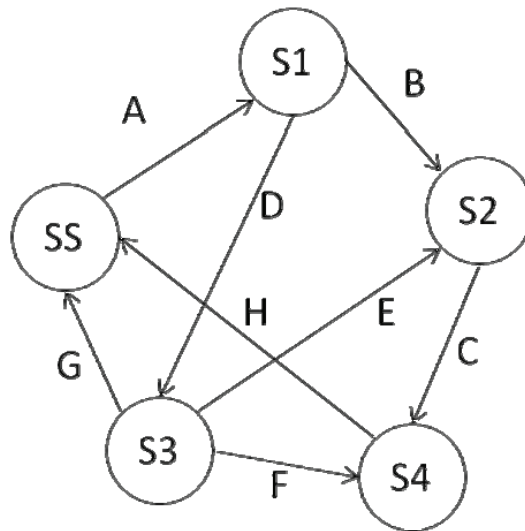
- i. Information about how the software is constructed.
- ii. Models of the system, software or components.
- iii. Analysis of the test basis documentation.
- iv. Analysis of the internal structure of the components.

Select ALL of the statements that describe the basis for black box techniques

- A** i and iv.
 - B** i and iii.
 - C** ii and iii.
 - D** ii and iv.
-

- 26** Which of the following is a benefit of carrying out static testing before dynamic testing
- A** It may find defects that would be more expensive to fix if they were found during dynamic testing.
 - B** It detects failures before release.
 - C** It is easier and takes less time to set up.
 - D** It would be impossible to find the same defects with dynamic testing.
-

27 Given the following state transition diagram:



Which of the test cases below will cover the following series of state transitions?

SS – S1 – S3 – S2 – S4

- A** A – D – G – H.
 - B** A – B – E – F.
 - C** A – D – E – C.
 - D** A – D – F – C.
-

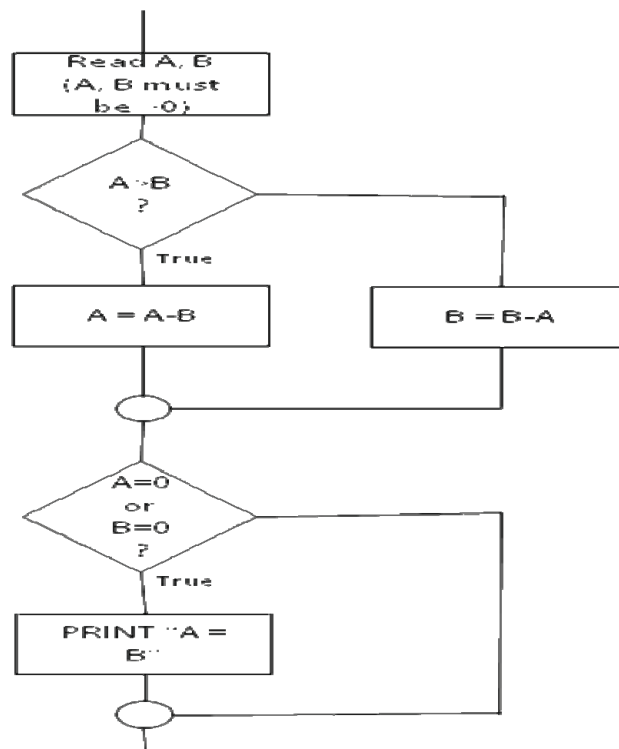
28 Given the following specification:

If you are less than 18, you are too young to be insured.
Between 18 and 30 inclusive, you will receive a 20% discount.
Anyone over 30 is not eligible for a discount.

Which of the following values for age are in the SAME equivalence partition?

- A** 29, 30, 31.
 - B** 17, 29, 31.
 - C** 18, 29, 30.
 - D** 17, 18, 19.
-

29 Given the following flow chart diagram:



What is the **minimum** number of test cases required for 100% statement coverage?

- A 2.
- B 1.
- C 3.
- D 4.

30 Given the following specification, which of the following values for age are in the **same** equivalence partition?

If you are under 14, you are too young to enter licensed premises.
Between 14 and 17 inclusive, you may enter licensed premises but not drink alcohol.
If you are 18 or over you may drink alcohol in moderation.

- A 14, 15, 16.
 - B 14, 17, 18.
 - C 13, 16, 17.
 - D 16, 17, 18.
-

31 What is a 'fault attack'?

- A** Error guessing based on typical failures in a given type of software.
 - B** Design of specific tests to address an enumerated list of possible failures.
 - C** Exploratory testing using aggressive test objectives.
 - D** Design of tests based on an analysis of defect detection rates.
-

32 What are key characteristics of a walkthrough?

- a) Led by the author, main purpose is learning
 - b) Led by a moderator, main purpose is defect finding
 - c) Led by the manager, main purpose is learning
 - d) Led by the author, main purpose is checking conformance to standards
-

33 Which of the following statements related to the selection of test techniques is true?

- A** Some techniques are more applicable to certain situations and test levels.
 - B** All techniques can be used in all situations and at all test levels.
 - C** Some techniques are not applicable at any test level.
 - D** There are no techniques that can be applied at the component testing level.
-

34 Given the following sample of pseudo code:

```
Input number_of_tracks_on_the_cd
Tracks_to_record = number_of_tracks_on_the_cd
Tracks_recorded = 0
While Tracks_to_record > 0
    Copy next track
    Tracks_to_record = Tracks_to_record - 1
    Tracks_recorded = Tracks_recorded + 1
End Loop
Print Tracks_recorded, "have been recorded".
```

What is the minimum number of test cases required to guarantee 100% decision coverage?

- A** 2.
 - B** 1.
 - C** 3.
 - D** 4.
-

35 A software house that build websites are having problems being raised by their customers relating to website navigation. Which of the following techniques are best suited to address this particular problem?

- i) equivalence partitioning
- ii) boundary value analysis
- iii) decision table testing
- iv) state transition testing
- v) exploratory testing
- vi) error guessing

- A** all of the above
 - B** ii, iii and iv
 - C** iv, v and vi
 - D** ii, iv and v
-

36 A system calculates the temperature of a room to the nearest 0.1 degree Celsius. When the room falls below 18 degrees the device displays “cold”, when the temperature rises to above 24 degrees it displays “hot”, and all other temperatures display “normal”.

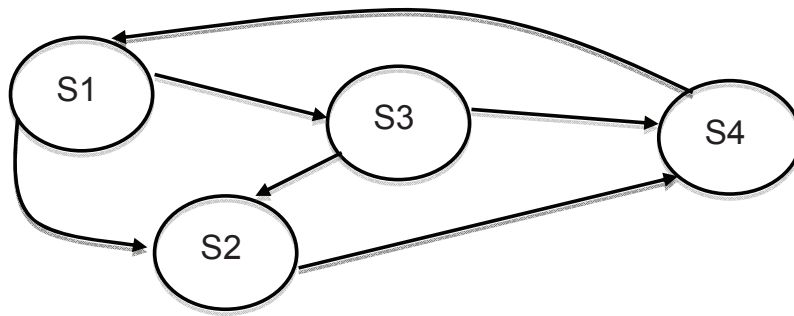
Using 2 value boundary value analysis technique what values should be chosen for your test cases?

- A** 17, 18, 24, 25
 - B** 17, 18, 19, 23, 24, 25
 - C** 18.0, 18.1, 23.9, 24.0
 - D** 17.9, 18.0, 24.0, 24.1
-

37 When would static analysis best be applied

- A** As soon as requirement specifications are produced
 - B** As soon as the code is produced but before compilation
 - C** As soon as the code is ready for system testing
 - D** As soon as the code is produced and after compilation but before any dynamic testing
-

38



Which of the following test cases achieve all states AND all transitions?

- A S1-S2-S3-S4
 - B S1-S3-S2-S4
 - C S1-S2-S4-S1-S3-S4-S1-S3-S2
 - D S1-S2-S4-S1-S3-S4
-

39 Which of the following test activities can be automated?

- i) Reviews and inspections.
- ii) Metrics gathering.
- iii) Test planning.
- iv) Test execution.
- v) Data generation.

- A i, iii, iv.
 - B ii, iii, v.
 - C i, ii, iii.
 - D ii, iv, v.
-

40 Which of the following tools is MOST likely to contain a comparator?

- A Dynamic Analysis tool.
- B Test Execution tool.
- C Static Analysis tool.
- D Security tool.

End of Paper

Commentary on the Questions - Sample Paper Day 2

Question 1

The correct answer is D. This is a K3 question.

The state matrix shows which events (the top row) affect which states (the left hand column). Row S2 has an entry in column E showing a transition to state S1; Row S3 has an entry in column F showing a transition to state S3 (i.e. a valid transition, but with no state change); Row S1 has an entry in column B, showing a transition to state S2. Row S3 has no entry in column E, so this would be an invalid transition and option D is therefore the correct answer.

The syllabus reference is section 4.3.4.

Question 2

The correct answer is A. This is a K2 question, though it is mostly a memory test.

The key piece of knowledge is that static testing does not execute code. Committing to memory which of the list is static and which is dynamic is one way of tackling this, but it is also possible to work out which are static. Inspections, being a type of review, are static. Data flow analysis is less obvious but the word analysis is a clue – unlike data flow testing, which would execute the code, data flow analysis simply analyses code without executing it. The correct choice of static techniques is therefore (iii) and (vi), and only one answer (A) has this combination. There are several relevant syllabus references but none directly answers the question (which is why it is a K2).

The syllabus references are section 3.1.3 and Section 4.

Question 3

The correct answer is D. This is a K1 question.

Specification-based test techniques are also called black box techniques; they are based on the specification or test basis documentation. Answer A is incorrect because it brings structure into the analysis; tests based on structure are white box or structural tests. Answer B is incorrect because it includes a requirement for experience, which points to experience-based techniques. Answer C is incorrect because it is related to structure-based (white box) testing. Answer D is correct because it is based on analysis rather than experience.

The syllabus reference is section 4.2.

Question 4

The correct answer is B. This is a K1 question.

Component testing and user acceptance testing are different test levels, so answer (a) is incorrect. Answer B is a true statement about component testing. Answer C could be true, though inspections would normally be used at the coding stage and would be a static test of the code before dynamic testing begins. Also, inspections are not always used. Answer D is a reference to the integration testing level. The choice is between answers B and C, and B is always true while C is sometimes true, so B is the correct answer.

The syllabus reference is section 2.2.1.

Question 5

The correct answer is A. This is a K2 question.

Answer B could have some validity but it is related to the design stage, which is well into the project life cycle, and it is a secondary effect rather than a direct influence on defect prevention. Answer C is attractive but testing cannot guarantee completeness of requirements or a “defect-free” status. Answer D has some attraction but the logic is flawed. Early test design would not necessarily familiarise testers with the design, neither would familiarity with the design necessarily enable them to test code more effectively; in any event testing of code will not make a significant impact on preventing defects, since the code will already have been written. Answer A is correct because it is a direct influence of testing on defect prevention at the most cost-effective opportunity.

The syllabus reference is section 1.2.

Question 6

The correct answer is B. This is a K3 question.

The code in the question contains a loop but no other decisions. To achieve 100% decision coverage we must exercise the loop decision ('while') in both the true and false direction. A test case that makes the 'while' true (e.g. a positive number of coins is input) will exercise the decision in the true direction and enter the loop. Each time the loop is exercised the statement `Number_of_Coins = Number_of_Coins - 1` will decrease the value of `Number_of_Coins`, so it will eventually reach zero, at which time the 'while' will take the false direction and the loop will terminate. Thus we can input any positive integer for `Number_of_Coins` and be confident that the decision will be exercised in both directions and 100% decision coverage will be achieved. One test case is enough, answer B.

The syllabus reference is Section 4.4.2.

Question 7

The correct answer is B. This is a K1 question.

This is a memory test. Impact analysis is the technique used to determine how a proposed change will affect a system; it does this by tracing the potential impact of the change in one area of a system to reach other parts of the structure (and hence determine how far a defect might propagate). Options A, C and D are all plausible and not easily eliminated unless you are confident of the correct answer.

The syllabus reference is section 2.4.

Question 8

The correct answer is B. This is a K1 question.

This question is determining whether you can distinguish between functional, non-functional and structural tests. Options A, C and D are all non-functional tests; only B is functional, i.e. it tests what the system can do rather than how it performs (how many new users, how many simultaneous users, how reliable).

The syllabus reference is sections 2.3.1/2.3.2.

Question 9

The correct answer is B. This is a K3 question.

This is a question that requires you to work through a decision table from the given inputs. Fred as a 32 year old smoker resident in London will trigger rule 4, giving an action of insure with no discount. This eliminates answers A and D. Jean-Michel, as a non-UK resident over 55 and a non-smoker will trigger rule 1, giving an action of do not insure. This eliminates answer c and leaves answer B as the correct answer.

The syllabus reference is section 4.3.3.

Question 10

The correct answer is A. This is a K1 question.

This question requires that you can remember the order of the phases of a formal review. Answer A is correct and there is no simple way to eliminate any of the others, since all begin with a phrase that could sound plausible if you were unsure of the correct order.

The syllabus reference is section 3.2.1.

Question 11

The correct answer is C This is a K2 question.

This question needs familiarity with the process of identifying and designing tests, which begins with analysis to determine test conditions, elaborates these into test cases which incorporate inputs and expected results, and then specifies the order of execution. The correct sequence begins with iii, so answer D can be eliminated. Next comes i, and this eliminates both A and B and leaves answer C as the correct answer.

The syllabus reference is section 4.1.

Question 12

The correct answer is B. This is a K3 question.

This question is similar to Question 19 and entails working through a decision table. In this question test case A exactly fits rule 1 and therefore generates an upgrade offer to First, which eliminates answers A and D. Test case D matches rule 4 and generates no upgrade offers, which makes answer B correct.

The syllabus reference is 4.3.3.

Question 13

The correct answer is D. This is a K1 question.

This is a recall question about static analysis. It is amenable to the following reasoning: static analysis is used to analyse code. Any of A, B or C could be detected by such analysis but D could not because it is being applied to a document that precedes code. In other words, even if you were not sure whether static analysis could do the things identified in A, B and C you could still reason that it could not do what is implied by D.

The syllabus reference is section 3.3.

Question 14

The correct answer is D. This is a K2 question.

This question requires an understanding of the nature of test automation, in particular the automation of test design. A test design tool is a tool to generate test designs, i.e. test cases. It does not generate test data; there is a separate tool for that purpose. A test design tool is also not a consistency checker, since this is a completely different function. The only options involving test case generation are iii and iv, so answer D is correct. Recognising that consistency checking is not part of test design would eliminate option i; recognising that test data preparation and test case design are different activities would eliminate option ii. The remaining two answers hinge on whether test design relates to specifications (black box design) or code (white box design). Both are test design activities and the tool category therefore relates to both.

The syllabus reference is section 6.1.4.

Question 15

The correct answer is B. This is a K3 question.

This is a question about both statement coverage and decision coverage. Knowing that decision coverage guarantees statement coverage we might usefully start by determining how many test cases are required to achieve decision coverage. We have two decision structures (IF...THEN...ELSE) with one embedded inside the other. This will require 3 test cases. One for outer decision false, one for outer decision true and inner decision true, one for outer decision true and inner decision false. This eliminates both A and D. The difference between B and C is whether we need 2 test cases for statement coverage or 3. Since every outcome of both branches has at least one statement in it we will have to exercise all of them, which will require 3 test cases, giving B as the correct answer.

The syllabus reference is 4.4.1/4.4.2.

Question 16

The correct answer is C. This is a K2 question.

This question requires a broad understanding of the review types. Answer a is correct but it is equally correct for walkthroughs. Answer b is also valid for both. Answer c is a valid differentiator, since other review types are normally led by someone other than the author. Answer d is true and this is distinct from the aims of walkthroughs, which are usually aimed more at education and finding defects. Answer e is equally true for both. Thus the options that imply difference are options c and d and this corresponds to answer C.

The syllabus reference is section 3.2.

Question 17

The correct answer is B. This is a K1 question.

This question can be approached by elimination if the correct answer is not immediately apparent. Answer A is incorrect because static analysis tools are not applied to requirements; identification of requirements inconsistencies would need a more sophisticated analysis. Answer C is incorrect because static analysis tools are not applied to specifications, which are text documents. Answer D is incorrect because incorrect input data cannot be identified statically; it depends on executing the code. (Additionally, incorrect data may not be a defect - the data may be incorrect, but the application behave correctly). This leaves answer B, which is a valid answer because a static analysis tool could check that module or component interfaces match. Answer B is therefore the correct answer.

The syllabus reference is section 3.3.

Question 18

The correct answer is D. This is a K3 question.

This is a question about boundary value analysis; it requires that we can recognise boundaries in the example given. Note that the question does not rely on any particular approach to boundary value analysis, i.e. the 2 value or 3 value approach. The simplest approach is to identify the partition boundaries from the code. These are at 4000, 5500 and 33500, so answers A, B and C are all incorrect and answer D is the correct answer. The distracters are all based on incorrect boundaries but they would be plausible to anyone who has not determined the correct boundaries in advance, so the best approach is to identify the correct boundaries before looking at the answers.

The syllabus reference is section 4.3.2.

Question 19

The correct answer is B. This is a K2 question.

This question requires an understanding of the relationships between test conditions, test cases and test data. Test conditions are derived from specifications and lead to test cases, which are then applied using test data. Thus test conditions are identified first, in test analysis; test cases are specified second, during test design; test data is also defined at the design stage, but after test cases; test scripts are specified when the tests are created during test implementation. This is answer B. An understanding of the basic sequence would help to eliminate answers A and D because neither relates test conditions to test analysis. Answer C can then be eliminated if it is recognised that test scripts are not created until test implementation.

The syllabus reference is sections 4.1.

Question 20

The correct answer is C. This is a K3 question.

This is an equivalence partition question based on the same stem as the last question. Having identified the boundaries before we can now identify the partitions, which will be 0 – 4000, 4001 – 5500, 5501 – 33500, and above 33500. This time we need to eliminate the options one by one.

Answer A has 2 members of the 5501 – 33500 partition so cannot be correct, answer B has 2 members of the 4001 – 5500 partition so cannot be correct, answer C has one member from each of 3 different partitions, so looks like the correct answer. Just to be sure we check answer D and find that it has 2 members from the above 33500 partition. So answer C is the correct answer.

The syllabus reference is section 4.3.1.

Question 21

The correct answer is C. This is a K3 question.

This question requires the ability to read a state table. Answer A represents a valid transition because the E4 column and the S3 row have an entry showing that state ES is reached via this transition. Answer B represents a valid transition because column E2 and state S1 meet at state S2. Answer C represents an invalid transition because there is no entry in the table at the junction of column E4 and state S2. Answer D represents a valid transition because column E3 and state SS meet at state S2.

The syllabus reference is section 4.3.

Question 22

The correct answer is A. This is a K3 question.

This question requires the ability to read and interpret a decision table. For test case A Jill holding the Foundation Certificate in row A eliminates Rule 1 and her not holding the Intermediate Certificate eliminates Rules 3 and 4. Rule 2 is therefore in force and she can neither enter the Practitioner exam nor be awarded a Practitioner Certificate. This eliminates answers B and C. In test case B Jack holds the Intermediate Certificate, which eliminates Rules 1 and 2, and has not taken his Practitioner exam, which eliminates Rule 4, so Rule 3 is in force. This eliminates answers C and D. Thus answer A is the only answer that is consistent with both test cases.

The syllabus reference is section 4.3.

Question 23

The correct answer is B. This is a K3 question.

This question requires the ability to identify equivalence partitions in a specification. The equivalence partitions in this question are 0-59, 60-79 and 80-100. Answer A includes two values from the second (pass) partition. Answer B includes one value from each partition. Answer C has two values from the second (pass) partition. Answer D includes two values from the third (distinction) partition. So answer B is the correct answer.

The syllabus reference is section 4.3.

Question 24

The correct answer is D. This is a K3 question.

This question requires the ability to read pseudo code. For statement 06 to be executed we require the decision at line 03 and the decision at line 05 both to be true. The decision at line 03 will be true if languages spoken is greater than 1 **and** number of countries visited is greater than 0. Answers A and B are therefore eliminated. The decision at line 06 will be true if the answer to the question at line 04 is 'Yes'. This eliminates answer C and leaves answer D as the correct answer.

The syllabus reference is section 4.4.

Question 25

The correct answer is C. This is a K2 question.

This is a question about the difference between black box and white box techniques. We know that black box techniques are specification-based and white box techniques are structure-based. Options i and iv are clearly structure related, so these are not correct. Option ii relates to models, which are alternatives to written specifications, and option iii relates to the test basis, which is the initial requirement specifications, so options ii and iii are correct and this points to answer C.

The syllabus reference is section 4.2.

Question 26

The correct answer is A. This is a K1 question.

Answer A is correct, but we can usefully cross check that all other answers are incorrect. Answer B is incorrect because static testing cannot detect failures since they occur when software is executed. Answer C is incorrect because the statement has no basis in theory or practice; it may be true sometimes but, on its own, it does not constitute an argument for static testing. Answer D is incorrect because some defects will be detectable by either static or dynamic testing; the advantage of static testing is the stage at which defects are discovered.

The syllabus reference is section 3.1.

Question 27

The correct answer is C. This is a K3 question.

This question requires the ability to read a state transition diagram. Transition SS to S1 requires event A; S1 to S3 requires event D (eliminating answers A and B); S3 to S2 requires event E (eliminating answer D); S2 to S4 requires event C, which confirms that answer C is correct because it is the only answer with the correct sequence of events (A, D, E, C).

The syllabus reference is section 4.4.

Sample Paper Day 2

Question 28

The correct answer is C. This is a K3 question.

This question is similar to question 37 but with a different scenario. This time we are looking for values in the same partition, but the approach is similar. First identify the partitions, which are 0 – 17, 18 – 30, and over 30.

Answer A is on the boundary of the middle partition (18 – 30) but has one value in the over 30 partition.

Answer B has 2 values in common with answer A and these are in two different partitions.

Answer C has all 3 values in the 18 – 30 partition, so looks correct.

Answer D has two values in the 0 - 17 partition and one in the 18 – 30 partition.

So, answer C is correct.

The syllabus reference is section 4.3.1.

Question 29

The correct answer is A. This is a K3 question.

This question requires the ability to read a flow chart and determine the test cases required to achieve a given level of coverage. For statement coverage the test cases must execute every line on the flow chart containing at least one statement. Thus the False exit from the second (bottom) decision does not need to be executed but the other three decision exits will need to be exercised to achieve statement coverage. The first decision requires two test cases to exercise its alternative exits. The second decision requires a single test case to exercise its true exit. Can the test case to exercise the second decision be one of those used to exercise the first decision? The answer is yes, because the inputs $A = B$ will make decision 1 false and decision two true. Thus two test cases are required, one with $A > B$ (say $A = 3, B = 2$) and one with $A = B$ (say both A and $B = 6$). Answer A is correct.

The syllabus references are sections 4.4.

Question 30

The correct answer is A. This is a K3 question.

This question is concerned with three partitions, where we are asked for values all in the same partition. The partitions are 0-13, 14-17, 18 upwards.

Answer A has three values all in the second partition. Answer B has two values in the second partition and one value in the third partition. Answer C has one value in the first partition and two in the second partition. Answer D has two values in the second partition and one in the third partition.

Answer A is correct.

The syllabus reference is section 4.4.

Question 31

The correct answer is B. This is a K1 question.

This question is recall based on an area of the syllabus updated in the 2010 version. Answer B is the based on the definition of fault attack given in the syllabus. There is no simple way to eliminate the other answers, so it is necessary to be able to recall the definition.

The syllabus reference is section 4.5.

Question 32

The correct answer is A. This is a K1 question

This question is a recall based upon section 3 of the syllabus where candidates need to be able to differentiate the differences between the 4 review types. The main purpose of a walkthrough is understanding and is led by the author.

The syllabus reference is section 3.2

Question 33

The correct answer is A. This is a K1 question.

This question requires a knowledge of where particular test techniques can be used. Answer B is incorrect because, for example, code-based techniques would not be appropriate at acceptance testing. Answer C is incorrect because this answer would make those techniques invalid. Answer D is incorrect because, for example, structure-based testing can be used at component level. Answer A is correct and based on the idea that techniques are selected to be appropriate at given test levels and situations.

The syllabus reference is section 4.6.

Sample Paper Day 2

Question 34

The correct answer is B. This is a K3 question.

This question requires the ability to read and interpret pseudo code and the ability to determine test cases needed for decision coverage. To achieve decision coverage all exits from all decisions must be exercised. In the pseudo code there is only one decision – the while loop entry decision. If the input is set up with `number_of_tracks_on_the_cd > 0` (say 2) then the while loop decision at line will be true and the loop will be entered. On entry to the loop the value of `Tracks_to_record` is reduced by 1 and the loop body terminates, which causes the entry decision to be re-evaluated. This value (`Tracks_to_record`) is now 1, so the loop is entered again. This time the loop decreases the value of `Tracks_to_record` to 0 and the loop body terminates and returns to re-evaluate the entry decision. `Tracks_to_record` is now 0, so the decision is false and the loop is not entered. In this single test case the entry decision has been exercised twice through the 'true' exit and once through the 'false' exit. The requirement for decision coverage is met and so decision coverage is achieved in 1 test case; answer B is the correct answer.

The syllabus reference is 4.4.

Question 35

The correct answer is C. This is a K2 question.

This question is based upon the application and use of test case design techniques in various contexts. The problems exist with web navigation and so the likely candidate is State Transition Testing. Unfortunately this option is in all of the potential answers. Exploratory Testing is a good approach to use since we can explore around this part of the system and produce charters for the navigational elements of the website. Error guessing is also a good suggestion as we can "guess" where errors might be.

The syllabus reference is 4.6

Question 36

The correct answer is D. This is a K3 question.

The normal partition is between 18.0 and 24.0. The precision is 0.1 and so we find ourselves with two further boundaries of 17.9 and 24.1.

The syllabus reference is 4.3.2

Question 37

The correct answer is B. This is a K2 question.

This question is based upon the knowledge of static analysis. Answer B is the best option as static analysis is performed on code and as soon as it is written. The distracter is answer D, however running static analysis before compilation might save more time.

The syllabus reference is 3.3

Question 38

The correct answer is C. This is a K3 question.

This question tests the candidate's knowledge of the application of State Transition Testing Technique. This test case covers all the circles (states) and all the lines (transitions).

The syllabus reference is 4.3.4

Question 39

The correct answer is D. This is a K2 question.

This question assesses the candidates knowledge of what activities test tools can support. Whilst many activities are supported with tools there are still some manual activities that need to be performed. In this question Reviews and Inspections must be performed manually (although tool support can assist with the logging and audit of the issues) and Test planning is a manual process. All other activities are supported by tools.

The syllabus reference is 6.1

Question 40

The correct answer is B. This is a K2 question.

This question checks to see if the candidate understands a particular activity of comparison, and which tool best supports this. The main tool would be a "comparison tool", however test execution tools (regression/capture replay tools) have comparison functions (albeit limited) within the tools

The syllabus reference is 6.1

End of Commentary

SAMPLE PAPER FOR DAY 3

Note: this paper has been created using questions from actual sample papers provided by BCS, the Chartered Institute for IT, with additional questions provided by Grove Consultants

Time allowed: 30 minutes

Record your surname/last/family name and initials on the Answer Sheet, writing in block capitals at the top and marking the relevant letter in each column.

Attempt all 20 multiple-choice questions – 1 mark awarded to each question.

A number of possible answers are given for each question, indicated by either **A. B. C.** or **D.** Only **one** of these answers and no more than one answer is correct in each multiple-choice question.

Your answers should be indicated on the Answer Sheet by making a **solid pencil mark** inside the box representing your chosen answer. These can be found to the right of the question number on your Answer Sheet.

If you make a mistake, erase your first mark, and put a mark in your new chosen answer column for that question.

Mark **only one answer** to each question.

Pass mark is 13/20

1 Which of the following would be a suitable metric for measuring achievement of progress towards a software quality target?

- A** Test coverage achieved at this test level.
 - B** Number of tests executed for this test level.
 - C** Number of tests planned for this test level.
 - D** Number of defects cleared at this test level.
-

2 Which of the following should be taken into account in deciding how much testing is required for an application?

- a. The level of business risk.
- b. The number of testers available.
- c. The project budget.
- d. The test tools available.
- e. The level of review skills.

- A** c and e.
 - B** b and d.
 - C** a and c.
 - D** a and d.
-

3 For testing, which of the options below BEST represents the main concerns of Configuration Management?

- i. All items of testware are identified and version controlled;
- ii. All items of testware are used in the final acceptance test;
- iii. All items of testware are stored in a common repository;
- iv. All items of testware are tracked for change;
- v. All items of testware are assigned to a responsible owner;
- vi. All items of testware are related to each other and to development items.

- A** ii, iii, v.
 - B** i, iii, iv.
 - C** iv, v, vi.
 - D** i, iv, vi.
-

4 Which of the following are valid objectives for incident reports?

- i. Provide developers and other parties with feedback about the problem to enable identification, isolation and correction as necessary.
- ii. Provide ideas for test process improvement.
- iii. Provide a vehicle for assessing tester competence.
- iv. Provide testers with a means of tracking the quality of the system under test.

- A** i, iii, iv.
 - B** i, ii, iii.
 - C** ii, iii, iv.
 - D** i, ii, iv.
-

5 In a PROACTIVE approach to testing when would you expect the bulk of the test design work to be started?

- A** During requirements analysis.
 - B** As early as possible.
 - C** After the software or system has been produced.
 - D** During development.
-

6 What is the MAIN purpose of a Master Test Plan?

- A** To communicate how testing will be performed.
 - B** To produce a test schedule.
 - C** To communicate how incidents will be managed.
 - D** To produce a work breakdown structure.
-

7 Which of the following lists of factors determines the effort required to test a software product?

- A** Characteristics of the product, the estimation process used and the outcome of testing.
 - B** Characteristics of the product, characteristics of the development process and the number of testers available.
 - C** Characteristics of the development process, number of testers available and the estimation process used.
 - D** Characteristics of the product, characteristics of the development process and the outcome of testing.
-

8 Which of the following is LEAST likely to be included in an incident report?

- A** Life cycle process in which the incident was discovered.
 - B** Suggestions for correcting the problem.
 - C** Date the incident was discovered.
 - D** Degree of impact on stakeholder interests.
-

- 9** What is the difference between a project risk and a product risk?
- A** Project risks are potential failure areas in the software or system; product risks are risks that surround the project's capability to deliver its objectives.
 - B** Project risks are risks that delivered software will not work; product risks are typically related to supplier issues, organizational factors and technical issues.
 - C** Project risks are typically related to supplier issues, organizational factors and technical issues; product risks are typically related to skill and staff shortages.
 - D** Project risks are the risks that surround the project's capability to deliver its objectives; product risks are potential failure areas in the software or system.
-

- 10** Which of the following is an objective of a pilot project for the introduction of a testing tool?
- A** Evaluate testers' competence to use the tool.
 - B** Assess whether the benefits will be achieved at reasonable cost.
 - C** Complete the testing of a key project.
 - D** Discover what the requirements for the tool are.
-

- 11** How is risk used to drive testing efforts?
- A** By ensuring that all requirements are met.
 - B** By ensuring that everything is tested.
 - C** By reducing the effect of supplier problems.
 - D** By determining where to start testing and where to test more.
-

- 12** Which of the following is determined by the amount of product risk identified?
- A** Size of the test team.
 - B** Scope for the use of test automation.
 - C** Levels of testing required.
 - D** Requirement for regression testing.
-

- 13** Which person would be the best to carry out the planning, monitoring and control of testing activities and tasks?
- A** A development manager with responsibility for producing the code to be tested.
 - B** Only the manager of a test group.
 - C** A quality assurance manager in a test leader role.
 - D** A project manager with responsibility for ensuring deadlines are met.
-

14 Which of the following would be a valid measure of test progress?

- A** Total number of defects in the product.
 - B** Effort required to fix all defects.
 - C** Number of test cases not yet executed.
 - D** Number of undetected defects.
-

15 Which of the following risks would testing be most effective in removing or reducing?

- A** The extent that requirements can be met given constraints such as time and cost.
 - B** Problems in defining the correct requirements.
 - C** Potential skill and staff shortages.
 - D** The potential that the software could cause harm to an individual.
-

16 At what stage in a software project should incident management be initiated?

- A** When requirements are used to generate the software design.
 - B** When requirements are captured and documented.
 - C** When the software is coded.
 - D** When testing of the software begins.
-

17 In a REACTIVE test approach when is the bulk of test design activity started?

- A** When the software specification is complete and design documentation is being produced.
 - B** At the beginning of the project when software requirements are being produced.
 - C** After the software has been produced.
 - D** After the user requirements have been documented.
-

18 What are **two** main functions of test reporting:

- a. To report when exit criteria were met for a given test level.
 - b. To identify when system testing will be complete.
 - c. To provide an analysis of metrics used to support decisions.
 - d. To define the test approach for the following test level.
 - e. To predict when entry criteria will be met for the next test level.
- A** b and d.
 - B** a and c.
 - C** c and e.
 - D** a and e.
-

- 19 Which of the following correctly describes possible impacts of tools on testing?
- A Tools can improve efficiency and reliability of testing but may also be intrusive.
 - B Tools can improve reliability of testing but cannot improve efficiency and they may be intrusive.
 - C Tools can improve efficiency of testing but cannot improve reliability and they may also be intrusive.
 - D Tools can improve efficiency and reliability of testing and are not intrusive.
-

- 20 How does configuration management **best** help to ensure the integrity and effectiveness of software testing?
- A By ensuring that traceability can be maintained throughout the test process.
 - B By ensuring that developers manage changes to code.
 - C By ensuring that requirements do not change.
 - D By ensuring that all specifications are under change control.
-

End of Paper

Commentary on the Questions - Sample Paper Day 3

Question 1

The correct answer is A. This is a K1 question.

This is a recall question but we could reasonably exclude answers B and C because they are related to test progress (progress against plan) rather than progress towards a quality target. Answer D is a quality-related measure, but not related to a quality target because the number of defects found and/or cleared at a test level is not a valid quality target.

The syllabus reference is section 5.3.

Question 2

The correct answer is C. This is a K2 question.

The question relates to how much testing is required, so it is not related to any limitations in resources to do the testing; these would affect how much testing could be done but not how much is required. In this way we can eliminate options b, d and e, leaving a and c as correct. Since A involves business risk we can be confident that this is a correct answer.

The syllabus reference is section 1.1.

Question 3

The correct answer is D. This is a K2 question.

All items of testware should be identified and version controlled, so (a) is correct, but it is not expected that all items should be used in final acceptance test, so (b) is incorrect. The idea of storing testware in a common repository is attractive but it is not required for configuration management, so (c) is incorrect. Items of testware must be tracked for change, so (d) is correct. Assignment of all testware to a responsible owner is also attractive but not a requirement of configuration management, so (e) is incorrect. Relating items of testware to each other and to development items is essential to ensure that any changes to development items can be identified for retest, so (f) is correct. With (a), (d) and (f) as the correct options, answer D is the correct answer.

The syllabus reference is section 5.4.

Question 4

The correct answer is D. This is a K2 question.

We are asked to select 3 valid objectives for an incident report from a list of 4 possibilities, so we could identify which of the 4 is not a valid objective and get to the answer with less work. Even if you are not sure about some of the possible objectives (all of which are listed in the syllabus) you may be able to eliminate option iii as an inappropriate use of incident management. In any event, answer D is correct.

The syllabus reference is section 5.6.

Question 5

The correct answer is B. This is a K1 question

This question requires an awareness of the alternative approaches to testing. In a proactive approach test design occurs before the software is available for testing.

The syllabus reference is section 5.2.5.

Question 6

The correct answer is A. This is a K1 question.

Whilst the syllabus covers all four of the options as planning activities, only A ('To communicate how testing will be performed') relates to the overall purpose. Other answers give detailed activities which, while correct, are not part of the main purpose (MAIN is capitalised in the question). Therefore A is the correct answer.

The syllabus reference is sections 5.2.1/5.2.2.

Question 7

The correct answer is D. This is a K2 question.

This question requires an understanding of what determines how much effort is required for testing. Note that some of the options relate to the time testing would take rather than the effort involved. For example, the number of testers might shorten testing but would not affect the total testing effort involved. On this basis we could reasonably eliminate answers B and C. The estimation process used may give a "better" idea of how much time will be required for testing, but in itself does not affect the time that will be actually used. This eliminates answer A, leaving answer D. This last mentioned details the three items that are given in the syllabus.

The syllabus reference is section 5.2.

Question 8

The correct answer is B. This is a K1 question.

The word **least** has here been emphasised because there is no absolute definition of what must be in an incident report. The question is really asking about the most important information for incident management. Options A, C and D are all important information related to prioritising and managing the incident. Option B is less significant, and this is signalled by the word 'suggestions'. The incident management process will ensure that the incident is assigned to a developer to identify the cause and correct the problem, so any suggestions for correction are optional rather than essential.

The syllabus reference is section 5.6.

Question 9

The correct answer is D. This is a K2 question.

The question offers variations on the definitions of product risk and project risk. Answer A swaps project risk for product risk in the definitions; answer B does the same thing but in a different way that focuses on the nature of project risks; answer C correctly characterises project risks but offers the wrong factors for product risks. Answer D offers the correct definitions of project risk and product risk.

The syllabus reference is sections 5.5.1/5.5.2.

Question 10

The correct answer is B This is a K1 question.

This question hinges on knowing that the purpose of a pilot project is to enable an organisation to decide whether a given tool meets their needs. Option A is not appropriate; staff will usually need training but this is not a barrier to acquisition of the tool. Option C is inappropriate for a pilot, which should be representative but definitely not 'key'. Option C is inappropriate because knowing the requirements was a prerequisite of selecting the tool for a pilot. Option B is the correct answer because it identifies whether the tool would make a worthwhile investment.

The syllabus reference is section 6.3.

Question 11

The correct answer is D This is a K1 question.

This question asks about the justification for a risk-based approach to testing. Options A and B are variations on the theme of being able to test everything or ensure that everything is OK and so relates to one of the key principles. Option C is related to a source of project risk rather than product risk. Option D is an expression of the idea that risk based testing prioritises testing and provides a basis for deciding when enough testing has been done.

The syllabus references are sections 1.3 and 5.5.

Question 12

The correct answer is C. This is a K1 question.

This is a question about how product risk drives testing. Recognising that risk drives the type and the extent of testing, it will determine what test levels are needed (answer C). Size of the test team may be affected by levels of risk but is determined by many other factors. Scope for test automation depends on the nature of the software being tested but is not directly affected by risk. A requirement for regression testing is triggered by changes to software.

The syllabus reference is section 5.5.2.

Question 13

The correct answer is C. This is a K1 question.

This question requires an understanding of the role of test leader. Answer A is not the most appropriate because the development manager would lack independence. Answer B is unnecessarily restrictive. Answer D is inappropriate because the drive to achieve deadlines could conflict with quality objectives. Answer C is the best answer because the quality assurance manager would be a good candidate for a test leader role and would have a measure of independence.

The syllabus reference is section 5.1.

Sample Paper Day 3

Question 14

The correct answer is C. This is a K1 question.

This is a question about test progress. In this question answers A and D are both related to factors that cannot be measured. Knowing either of these would make testing much easier! Answer B is also implicitly unmeasurable; since we cannot know how many defects are yet to be found we cannot determine how much effort it would take to correct them. Thus C is the correct answer and we can see that it is a simple measurement of the test cases planned but not yet executed.

The syllabus reference is section 5.3.1.

Question 15

The correct answer is D. This is a K1 question.

This question is based on the definition of a product risk. Risk-based testing is defined as that which is used to address product risks. Answer D is the correct definition as a matter of recall. Answers A, B and C all relate to project risks rather than product risks.

The syllabus reference is section 5.5.

Question 16

The correct answer is B. This is a K1 question.

This question requires recall of the need for incident management to be initiated at the earliest possible opportunity. The four answers could be placed in time sequence as B, A, C, D. Thus B is the correct answer.

The syllabus reference is section 5.6.

Question 17

The correct answer is C. This is a K1 question.

This question requires an awareness of the alternative approaches to testing. Answer C is correct since a reactive approach to test design comes after the software or system has been produced (i.e. test design is started as a reaction to the software under test becoming available for test execution). Alternatively, since 'reactive' suggests 'after some event', we could reasonably suggest that B is not correct and, since answers A and D also imply early testing, they are likely to be less 'reactive' than answer C.

The syllabus reference is section 5.2.

Question 18

The correct answer is B. This is a K2 question.

This question requires the ability to separate test reporting activities from other activities in the fundamental test process. Option a actually uses 'report' and looks like a good candidate. Option b is about evaluating exit criteria at a specific test level, so is not about reporting in general. This eliminates answer A. Option c 'providing an analysis' is a reporting activity, so this looks like a good candidate. Option d is about definition of test approach, so planning. Option e is about estimation and planning. This eliminates answers C and D. B is the correct answer because it combines our two valid options.

The syllabus reference is section 5.3.

Question 19

The correct answer is A. This is a K2 question.

This question combines three important characteristics of software testing tools: efficiency, reliability and intrusiveness. Clearly tools can improve efficiency (e.g. automating regression testing) and reliability (e.g. tests can be repeated multiple times without error), but they may intrude (e.g. by instrumenting code to enable coverage to be measured). Thus answer A is correct and the other answers simply offer three other combinations of the characteristics.

The syllabus reference is section 6.1.

Question 20

The correct answer is A. This is a K1 question.

This question requires an understanding of the role and purpose of configuration management for testing. Answer A is correct and traceability is essential to the integrity and effectiveness of testing, so this looks like a good candidate. Answers B and D are attractive because control of code and specifications can help prevent a chaotic environment which would make testing more difficult, but they do not ensure traceability or that test deliverables are correctly related to development deliverables. Answer C is incorrect because configuration management cannot prevent requirements changes. Answer A is the most inclusive, and therefore the best answer.

The syllabus reference is 5.4.

End of Commentary

Extra Exam Questions

Contents

- 1 Fundamentals of Testing
- 2 Testing in the Lifecycle
- 3 Static Testing Techniques
- 4 Test Design Techniques
- 5 Test Management
- 6 Tool Support in Testing

Question: 1-1

A company recently bought a 3rd party package and plan to run acceptance tests against it. Which is of the following is the most likely test objective?

- a) to build confidence in the application
- b) to detect bugs in the application
- c) to train the users
- d) to show that the other company has not done enough testing

Question: 1-2

Ensuring that test design starts during the requirements definition phase is important to enable which of the following test objectives?

- a) gain confidence in the system
- b) find defects through dynamic testing
- c) prevent defects in the system
- d) finishing the project on time

Question: 1-3

A system test group finds 95% of the defects in a project prior to implementation. Which of the following testing principles is likely to help the test manager explain why some defects have been missed?

- a) pesticide paradox
- b) defect clustering
- c) exhaustive testing is impossible
- d) absence-of-errors fallacy

Question: 1-4

A test team is repeating the same tests and not finding any new defects. Which of the following testing principles is likely to help the test manager understand why this is happening?

- a) pesticide paradox
- b) defect clustering
- c) exhaustive testing is impossible
- d) absence-of-errors fallacy

Question: 1-5

The word “bug” is synonymous with which of the following?

- a) incident
- b) mistake
- c) error
- d) defect

Question: 1-6

Which activity in the fundamental test process involves producing the test procedure specification?

- a) planning and control
- b) analysis and design
- c) implementation and execution
- d) evaluating exit criteria and reporting

Question: 1-7

Which of the following activities about testing and debugging are true/false.

- i. developers perform debugging at any level of testing
- ii. developers raise incident reports on failures they generate
- iii. testing is only done by testers
- iv. re-testing (confirmation testing) is only done by testers
- v. testers repair defects in the code
- vi. regression testing is a testing activity

- a) i, ii and iii are TRUE; iv, v and vi are FALSE
- b) ii, iii and v are TRUE; i, iv and vi are FALSE
- c) i and vi are TRUE, ii, iii, iv, v are FALSE
- d) v and vi are TRUE; i, ii, iii and iv are FALSE

2 Testing in the Lifecycle

Question: 2-1

What lifecycle model best describes having sequential analysis and test stages?

- a) iterative model
- b) v-model
- c) agile model
- d) incremental model

Question: 2-2

Which of the following is not a test type (target)?

- a) functional testing
- b) testing changes to the system
- c) performance testing
- d) component testing

Question: 2-3

Which of the following is not an aspect of maintenance testing?

- a) testing modifications
- b) testing retirement
- c) testing maintainability
- d) testing migration

Question: 2-4

A regression test:

- a) will always be automated
- b) is run every week
- c) will check unchanged areas of the system have not been affected by changes being made
- d) will check changed areas of the system have not been affected by the changes being made

Question: 2-5

Which test target best describes the following test: “checking how easy the system is to use”?

- a) functional
- b) non-functional
- c) structural
- d) changes

Question: 2-6

Which of the following statements is true? Select ALL correct options

Regression testing should be performed:

- i once a month
 - ii when a defect has been fixed
 - iii when the test environment has changed
 - iv when the software has been changed
- a) II and IV
 - b) II, III and IV
 - c) I, II and III
 - d) I and III

Question: 2-7

Which of the following is NOT a non-functional characteristic test target

- a) usability testing
- b) performance testing
- c) feasibility testing
- d) reliability testing

Question: 2-8

Which of the following is NOT a type of acceptance test?

- a) user
- b) maintenance
- c) alpha
- d) operational

Question: 2-9

What is the main purpose of regression testing?

- a) to verify the success of corrective actions
- b) to prevent a task from being incorrectly considered completed
- c) to ensure defects have not been introduced by a modification
- d) to encourage better unit testing by the programmers

Question: 2-10

What is the main purpose of re-testing (confirmation testing)?

- a) to verify the success of corrective actions
- b) to prevent a task from being incorrectly considered completed
- c) to ensure defects have not been introduced by a modification
- d) to encourage better unit testing by the programmers

3 Static Testing Techniques

Question: 3-1

Which of the following artefacts can be examined by using review techniques?

- a) software code
- b) requirement specifications
- c) test cases
- d) all of the above

Question: 3-2

Which type of review has the main purpose of education?

- a) walkthroughs
- b) informal reviews
- c) technical reviews
- d) inspections

Question: 3-3

Which of the following statements about inspections is false?

- a) inspections are led by a trained moderator
- b) documents need to be written down
- c) source documents are issued
- d) inspections are led by a trained author

Question: 3-4

Which statement about static analysis tools is true?

- a) they can detect memory leaks
- b) they give information about which parts of the code has been covered
- c) they provide information about the quality of the code without executing it
- d) they compare actual results and expected results

Question: 3-5

Which fault cannot be found using a static analysis tool?

- a) dead code
- b) wrong requirements being specified
- c) infinite loops
- d) variables re-defined without being used

4 Test Design Techniques

Question: 4-1

Which of the following is not a black box technique?

- a) decision tables
- b) boundary value analysis
- c) decision testing
- d) use cases

Question: 4-2

What is a key characteristic of white box testing?

- a) they are mainly used to assess the structure of the specification
- b) they are based on the skills and experience of the tester
- c) they use a formal or informal model of the software component
- d) they are used both to measure coverage and to design tests to increase coverage

Question: 4-3

What does it mean if a set of tests has achieved 90% decision coverage?

- a) 9 out of 10 tests have been run on this software
- b) 9 out of 10 decision outcomes have been exercised by this set of tests
- c) 9 out of 10 statements have been exercised by this set of tests
- d) 9 out of 10 business rules have been exercised by this set of tests

Question: 4-4

How many tests are needed to achieve 100% statement coverage?

```
Read(A)
IF A < 10 THEN
    B = A
ELSE
    IF A > 10 THEN
        B = 10
    ENDIF
ENDIF
ENDIF
```

- a) 1
- b) 2
- c) 3
- d) 4

Question: 4-5

How many tests are needed to achieve 100% decision coverage?

```
Read(A)
IF A < 10 THEN
    B = A
ELSE
    IF A > 10 THEN
        B = 10
    ENDIF
ENDIF
```

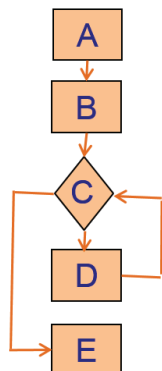
- a) 1
- b) 2
- c) 3
- d) 4

Question: 4-6

Test A covers path: A, B, C, D, C, E

Test B covers path: A, B, C, E

Which of the following statements below is correct?



- a) test A and B cover the same amount of decision coverage
- b) test A and B cover the same amount of statement coverage
- c) test A covers 100% decision coverage
- d) test B is a more efficient test

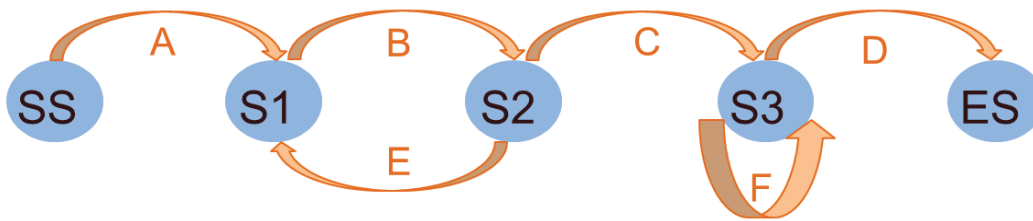
Question: 4-7

If the temperature of the room falls below 18 degrees the heating is switched on. When the temperature of the room reaches 21 degrees, the heating is switched off. What is the minimum set of test input values to cover all valid equivalence partitions?

- a) 15, 19, 25
- b) 17, 18, 19 and 20
- c) 18, 20 and 22
- d) 16 and 26

Question: 4-8

Given the following state transition diagram



which of the following series of state transitions below will provide transition coverage?

- a) A, B, E, B, C, F, D
- b) A, B, E, B, C, F, F
- c) A, B, E, B, C, D
- d) A, B, C, F, F, D

Question: 4-9

Which type of test design technique typically uses a “charter” and is useful when you have limited time and limited or no specification

- a) Exploratory testing
- b) State Transition testing
- c) Error guessing
- d) Decision testing

Question: 4-10

Which of the following is NOT a dynamic test technique

- a) Statement testing
- b) State transition testing
- c) Error guessing
- d) Walkthroughs

Question: 4-11

Which is the correct order in generating tests

- a) Test conditions, analyse test basis, test cases, test procedures
- b) Analyse test basis, test cases, test conditions, test procedures
- c) Analyse test basis, test conditions, test cases, test procedures
- d) Test conditions, test cases, test procedures, analyse test basis

5 Test Management

Question: 5-1

Which of the following is not part of the test plan according to IEEE829?

- a) schedule
- b) risks and contingencies
- c) quality plan
- d) suspension criteria

Question: 5-2

You are writing a test plan using IEEE829 template and are currently completing the risks and contingencies section. Which of the following is likely to be listed as a project risk?

- a) potentially slow transaction timings
- b) unexpected illness of a team member
- c) incorrect calculations
- d) 3rd party software received with poor quality

Question: 5-3

Which of the following is among the typical tasks of a test leader?

- a) develop system requirements, design specifications and usage models
- b) run automation tests
- c) produce test specifications
- d) gather and report test progress metrics

Question: 5-4

A potential benefit of independent testing is:

- a) developers do not have to test their code
- b) testers will find more defects
- c) a different view is taken which may lead to more defects being found
- d) different levels of testing is implemented within the organisation

Question: 5-5

Which is the MOST important advantage of independence in testing?

- a) An independent tester may find defects more quickly than the person who wrote the software.
- b) An independent tester may be more focused on showing how the software works than the person who wrote the software
- c) An independent tester may be more effective and efficient because they are less familiar with the software than the person who wrote it
- d) An independent tester may be more effective at finding defects missed by the person who wrote the software

6 Tool Support in Testing

Question: 6-1

Which type of tool is best for comparing files, reports and screens?

- a) test execution tool
- b) test comparator
- c) test management tool
- d) test data preparation tool

Question: 6-2

Which type of tool is best to find memory leaks and resource allocation?

- a) dynamic analysis
- b) modelling tools
- c) monitoring tools
- d) security tools

Question: 6-3

Which type of tool is mainly used by developers?

- a) performance tool
- b) coverage tool
- c) test design tool
- d) configuration management tool

Question: 6-4

Which type of tool will provide significant benefits if a data-driven or keyword driven approach to scripting is used?

- a) test management tool
- b) debugging tool
- c) test execution tool
- d) performance tool

Question: 6-5

Which type of tool is not strictly a testing tool but helps testing by continuously analysing resources and is mainly used by operations?

- a) performance tools
- b) dynamic analysis tools
- c) configuration management tools
- d) monitoring tools

Question: 6-6

Which type of tool can be intrusive?

- a) dynamic analysis tool
- b) performance testing tool
- c) coverage measurement tool
- d) static analysis tool

Question: 6-7

Which of the following are disadvantages of capturing tests by recording the actions of a manual tester?

- I The script may be unstable when unexpected events occur
- II Data for a number of similar tests is automatically stored separately from the script
- III Expected results must be added to the captured script
- IV The captured script documents the exact inputs entered by the tester
- V When replaying a captured test, the tester may need to debug the script if it doesn't play correctly

- a) I, III, IV and V
- b) II, IV and V
- c) I, II and IV
- d) I and V

Extra Exam Answer List

Answers:

1-1	A	2-1	B	3-1	D	4-1	C	5-1	C	6-1	B
1-2	C	2-2	D	3-2	A	4-2	D	5-2	B	6-2	A
1-3	C	2-3	C	3-3	D	4-3	B	5-3	D	6-3	B
1-4	A	2-4	C	3-4	C	4-4	B	5-4	C	6-4	C
1-5	D	2-5	B	3-5	B	4-5	C	5-5	D	6-5	D
1-6	C	2-6	B			4-6	C			6-6	C
1-7	C	2-7	C			4-7	A			6-7	A
		2-8	B			4-8	A				
		2-9	C			4-9	A				
		2-10	A			4-10	D				
						4-11	C				

Table of Contents

Acknowledgements.....	7
Introduction to this Syllabus.....	8
Purpose of this Document.....	8
The Certified Tester Foundation Level in Software Testing.....	8
Learning Objectives/Cognitive Level of Knowledge.....	8
The Examination.....	8
Accreditation.....	8
Level of Detail.....	9
How this Syllabus is Organized.....	9
1. Fundamentals of Testing (K2).....	10
1.1 Why is Testing Necessary (K2).....	11
1.1.1 Software Systems Context (K1).....	11
1.1.2 Causes of Software Defects (K2).....	11
1.1.3 Role of Testing in Software Development, Maintenance and Operations (K2).....	11
1.1.4 Testing and Quality (K2).....	11
1.1.5 How Much Testing is Enough? (K2).....	12
1.2 What is Testing? (K2).....	13
1.3 Seven Testing Principles (K2).....	14
1.4 Fundamental Test Process (K1).....	15
1.4.1 Test Planning and Control (K1).....	15
1.4.2 Test Analysis and Design (K1).....	15
1.4.3 Test Implementation and Execution (K1).....	16
1.4.4 Evaluating Exit Criteria and Reporting (K1).....	16
1.4.5 Test Closure Activities (K1).....	16
1.5 The Psychology of Testing (K2).....	18
1.6 Code of Ethics.....	20
2. Testing Throughout the Software Life Cycle (K2).....	21
2.1 Software Development Models (K2).....	22
2.1.1 V-model (Sequential Development Model) (K2).....	22
2.1.2 Iterative-incremental Development Models (K2).....	22
2.1.3 Testing within a Life Cycle Model (K2).....	22
2.2 Test Levels (K2).....	24
2.2.1 Component Testing (K2).....	24
2.2.2 Integration Testing (K2).....	25
2.2.3 System Testing (K2).....	26
2.2.4 Acceptance Testing (K2).....	26
2.3 Test Types (K2).....	28
2.3.1 Testing of Function (Functional Testing) (K2).....	28
2.3.2 Testing of Non-functional Software Characteristics (Non-functional Testing) (K2).....	28
2.3.3 Testing of Software Structure/Architecture (Structural Testing) (K2).....	29
2.3.4 Testing Related to Changes: Re-testing and Regression Testing (K2).....	29
2.4 Maintenance Testing (K2).....	30
3. Static Techniques (K2).....	31
3.1 Static Techniques and the Test Process (K2).....	32
3.2 Review Process (K2).....	33
3.2.1 Activities of a Formal Review (K1).....	33
3.2.2 Roles and Responsibilities (K1).....	33
3.2.3 Types of Reviews (K2).....	34
3.2.4 Success Factors for Reviews (K2).....	35
3.3 Static Analysis by Tools (K2).....	36
4. Test Design Techniques (K4).....	37
4.1 The Test Development Process (K3).....	38
4.2 Categories of Test Design Techniques (K2).....	39

4.3	Specification-based or Black-box Techniques (K3)	40
4.3.1	Equivalence Partitioning (K3)	40
4.3.2	Boundary Value Analysis (K3)	40
4.3.3	Decision Table Testing (K3)	40
4.3.4	State Transition Testing (K3)	41
4.3.5	Use Case Testing (K2)	41
4.4	Structure-based or White-box Techniques (K4)	42
4.4.1	Statement Testing and Coverage (K4)	42
4.4.2	Decision Testing and Coverage (K4)	42
4.4.3	Other Structure-based Techniques (K1)	42
4.5	Experience-based Techniques (K2)	43
4.6	Choosing Test Techniques (K2)	44
5.	Test Management (K3)	45
5.1	Test Organization (K2)	47
5.1.1	Test Organization and Independence (K2)	47
5.1.2	Tasks of the Test Leader and Tester (K1)	47
5.2	Test Planning and Estimation (K3)	49
5.2.1	Test Planning (K2)	49
5.2.2	Test Planning Activities (K3)	49
5.2.3	Entry Criteria (K2)	49
5.2.4	Exit Criteria (K2)	49
5.2.5	Test Estimation (K2)	50
5.2.6	Test Strategy, Test Approach (K2)	50
5.3	Test Progress Monitoring and Control (K2)	51
5.3.1	Test Progress Monitoring (K1)	51
5.3.2	Test Reporting (K2)	51
5.3.3	Test Control (K2)	51
5.4	Configuration Management (K2)	52
5.5	Risk and Testing (K2)	53
5.5.1	Project Risks (K2)	53
5.5.2	Product Risks (K2)	53
5.6	Incident Management (K3)	55
6.	Tool Support for Testing (K2)	57
6.1	Types of Test Tools (K2)	58
6.1.1	Tool Support for Testing (K2)	58
6.1.2	Test Tool Classification (K2)	58
6.1.3	Tool Support for Management of Testing and Tests (K1)	59
6.1.4	Tool Support for Static Testing (K1)	59
6.1.5	Tool Support for Test Specification (K1)	59
6.1.6	Tool Support for Test Execution and Logging (K1)	60
6.1.7	Tool Support for Performance and Monitoring (K1)	60
6.1.8	Tool Support for Specific Testing Needs (K1)	60
6.2	Effective Use of Tools: Potential Benefits and Risks (K2)	62
6.2.1	Potential Benefits and Risks of Tool Support for Testing (for all tools) (K2)	62
6.2.2	Special Considerations for Some Types of Tools (K1)	62
6.3	Introducing a Tool into an Organization (K1)	64
7.	References	65
	Standards	65
	Books	65
8.	Appendix A – Syllabus Background	67
	History of this Document	67
	Objectives of the Foundation Certificate Qualification	67
	Objectives of the International Qualification (adapted from ISTQB meeting at Sollentuna, November 2001)	67
	Entry Requirements for this Qualification	67

Background and History of the Foundation Certificate in Software Testing	68
9. Appendix B – Learning Objectives/Cognitive Level of Knowledge	69
Level 1: Remember (K1)	69
Level 2: Understand (K2)	69
Level 3: Apply (K3)	69
Level 4: Analyze (K4)	69
10. Appendix C – Rules Applied to the ISTQB	71
Foundation Syllabus	71
10.1.1 General Rules	71
10.1.2 Current Content	71
10.1.3 Learning Objectives	71
10.1.4 Overall Structure	71
11. Appendix D – Notice to Training Providers	73
12. Appendix E – Release Notes	74
Release 2010	74
Release 2011	74
13. Index	76

Introduction to this Syllabus

Purpose of this Document

This syllabus forms the basis for the International Software Testing Qualification at the Foundation Level. The International Software Testing Qualifications Board (ISTQB) provides it to the National Boards for them to accredit the training providers and to derive examination questions in their local language. Training providers will determine appropriate teaching methods and produce courseware for accreditation. The syllabus will help candidates in their preparation for the examination. Information on the history and background of the syllabus can be found in Appendix A.

The Certified Tester Foundation Level in Software Testing

The Foundation Level qualification is aimed at anyone involved in software testing. This includes people in roles such as testers, test analysts, test engineers, test consultants, test managers, user acceptance testers and software developers. This Foundation Level qualification is also appropriate for anyone who wants a basic understanding of software testing, such as project managers, quality managers, software development managers, business analysts, IT directors and management consultants. Holders of the Foundation Certificate will be able to go on to a higher-level software testing qualification.

Learning Objectives/Cognitive Level of Knowledge

Learning objectives are indicated for each section in this syllabus and classified as follows:

- K1: remember
- K2: understand
- K3: apply
- K4: analyze

Further details and examples of learning objectives are given in Appendix B.

All terms listed under “Terms” just below chapter headings shall be remembered (K1), even if not explicitly mentioned in the learning objectives.

The Examination

The Foundation Level Certificate examination will be based on this syllabus. Answers to examination questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable.

The format of the examination is multiple choice.

Exams may be taken as part of an accredited training course or taken independently (e.g., at an examination center or in a public exam). Completion of an accredited training course is not a pre-requisite for the exam.

Accreditation

An ISTQB National Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus, and is allowed to have an ISTQB examination as part of the course.

Further guidance for training providers is given in Appendix D.

Level of Detail

The level of detail in this syllabus allows internationally consistent teaching and examination. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Foundation Level
- A list of information to teach, including a description, and references to additional sources if required
- Learning objectives for each knowledge area, describing the cognitive learning outcome and mindset to be achieved
- A list of terms that students must be able to recall and understand
- A description of the key concepts to teach, including sources such as accepted literature or standards

The syllabus content is not a description of the entire knowledge area of software testing; it reflects the level of detail to be covered in Foundation Level training courses.

How this Syllabus is Organized

There are six major chapters. The top-level heading for each chapter shows the highest level of learning objectives that is covered within the chapter and specifies the time for the chapter. For example:

2. Testing Throughout the Software Life Cycle (K2) 115 minutes

This heading shows that Chapter 2 has learning objectives of K1 (assumed when a higher level is shown) and K2 (but not K3), and it is intended to take 115 minutes to teach the material in the chapter. Within each chapter there are a number of sections. Each section also has the learning objectives and the amount of time required. Subsections that do not have a time given are included within the time for the section.

1. Fundamentals of Testing (K2)

155 minutes

Learning Objectives for Fundamentals of Testing

The objectives identify what you will be able to do following the completion of each module.

1.1 Why is Testing Necessary? (K2)

- LO-1.1.1 Describe, with examples, the way in which a defect in software can cause harm to a person, to the environment or to a company (K2)
- LO-1.1.2 Distinguish between the root cause of a defect and its effects (K2)
- LO-1.1.3 Give reasons why testing is necessary by giving examples (K2)
- LO-1.1.4 Describe why testing is part of quality assurance and give examples of how testing contributes to higher quality (K2)
- LO-1.1.5 Explain and compare the terms error, defect, fault, failure, and the corresponding terms mistake and bug, using examples (K2)

1.2 What is Testing? (K2)

- LO-1.2.1 Recall the common objectives of testing (K1)
- LO-1.2.2 Provide examples for the objectives of testing in different phases of the software life cycle (K2)
- LO-1.2.3 Differentiate testing from debugging (K2)

1.3 Seven Testing Principles (K2)

- LO-1.3.1 Explain the seven principles in testing (K2)

1.4 Fundamental Test Process (K1)

- LO-1.4.1 Recall the five fundamental test activities and respective tasks from planning to closure (K1)

1.5 The Psychology of Testing (K2)

- LO-1.5.1 Recall the psychological factors that influence the success of testing (K1)
- LO-1.5.2 Contrast the mindset of a tester and of a developer (K2)

1.1 Why is Testing Necessary (K2)

20 minutes

Terms

Bug, defect, error, failure, fault, mistake, quality, risk

1.1.1 Software Systems Context (K1)

Software systems are an integral part of life, from business applications (e.g., banking) to consumer products (e.g., cars). Most people have had an experience with software that did not work as expected. Software that does not work correctly can lead to many problems, including loss of money, time or business reputation, and could even cause injury or death.

1.1.2 Causes of Software Defects (K2)

A human being can make an error (mistake), which produces a defect (fault, bug) in the program code, or in a document. If a defect in code is executed, the system may fail to do what it should do (or do something it shouldn't), causing a failure. Defects in software, systems or documents may result in failures, but not all defects do so.

Defects occur because human beings are fallible and because there is time pressure, complex code, complexity of infrastructure, changing technologies, and/or many system interactions.

Failures can be caused by environmental conditions as well. For example, radiation, magnetism, electronic fields, and pollution can cause faults in firmware or influence the execution of software by changing the hardware conditions.

1.1.3 Role of Testing in Software Development, Maintenance and Operations (K2)

Rigorous testing of systems and documentation can help to reduce the risk of problems occurring during operation and contribute to the quality of the software system, if the defects found are corrected before the system is released for operational use.

Software testing may also be required to meet contractual or legal requirements, or industry-specific standards.

1.1.4 Testing and Quality (K2)

With the help of testing, it is possible to measure the quality of software in terms of defects found, for both functional and non-functional software requirements and characteristics (e.g., reliability, usability, efficiency, maintainability and portability). For more information on non-functional testing see Chapter 2; for more information on software characteristics see 'Software Engineering – Software Product Quality' (ISO 9126).

Testing can give confidence in the quality of the software if it finds few or no defects. A properly designed test that passes reduces the overall level of risk in a system. When testing does find defects, the quality of the software system increases when those defects are fixed.

Lessons should be learned from previous projects. By understanding the root causes of defects found in other projects, processes can be improved, which in turn should prevent those defects from reoccurring and, as a consequence, improve the quality of future systems. This is an aspect of quality assurance.

Testing should be integrated as one of the quality assurance activities (i.e., alongside development standards, training and defect analysis).

1.1.5 How Much Testing is Enough? (K2)

Deciding how much testing is enough should take account of the level of risk, including technical, safety, and business risks, and project constraints such as time and budget. Risk is discussed further in Chapter 5.

Testing should provide sufficient information to stakeholders to make informed decisions about the release of the software or system being tested, for the next development step or handover to customers.

1.2 What is Testing? (K2)

30 minutes

Terms

Debugging, requirement, review, test case, testing, test objective

Background

A common perception of testing is that it only consists of running tests, i.e., executing the software. This is part of testing, but not all of the testing activities.

Test activities exist before and after test execution. These activities include planning and control, choosing test conditions, designing and executing test cases, checking results, evaluating exit criteria, reporting on the testing process and system under test, and finalizing or completing closure activities after a test phase has been completed. Testing also includes reviewing documents (including source code) and conducting static analysis.

Both dynamic testing and static testing can be used as a means for achieving similar objectives, and will provide information that can be used to improve both the system being tested and the development and testing processes.

Testing can have the following objectives:

- Finding defects
- Gaining confidence about the level of quality
- Providing information for decision-making
- Preventing defects

The thought process and activities involved in designing tests early in the life cycle (verifying the test basis via test design) can help to prevent defects from being introduced into code. Reviews of documents (e.g., requirements) and the identification and resolution of issues also help to prevent defects appearing in the code.

Different viewpoints in testing take different objectives into account. For example, in development testing (e.g., component, integration and system testing), the main objective may be to cause as many failures as possible so that defects in the software are identified and can be fixed. In acceptance testing, the main objective may be to confirm that the system works as expected, to gain confidence that it has met the requirements. In some cases the main objective of testing may be to assess the quality of the software (with no intention of fixing defects), to give information to stakeholders of the risk of releasing the system at a given time. Maintenance testing often includes testing that no new defects have been introduced during development of the changes. During operational testing, the main objective may be to assess system characteristics such as reliability or availability.

Debugging and testing are different. Dynamic testing can show failures that are caused by defects. Debugging is the development activity that finds, analyzes and removes the cause of the failure. Subsequent re-testing by a tester ensures that the fix does indeed resolve the failure. The responsibility for these activities is usually testers test and developers debug.

The process of testing and the testing activities are explained in Section 1.4.

1.3 Seven Testing Principles (K2)

35 minutes

Terms

Exhaustive testing

Principles

A number of testing principles have been suggested over the past 40 years and offer general guidelines common for all testing.

Principle 1 – Testing shows presence of defects

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

Principle 2 – Exhaustive testing is impossible

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.

Principle 3 – Early testing

To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives.

Principle 4 – Defect clustering

Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.

Principle 5 – Pesticide paradox

If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this “pesticide paradox”, test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.

Principle 6 – Testing is context dependent

Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.

Principle 7 – Absence-of-errors fallacy

Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.

1.4 Fundamental Test Process (K1)

35 minutes

Terms

Confirmation testing, re-testing, exit criteria, incident, regression testing, test basis, test condition, test coverage, test data, test execution, test log, test plan, test procedure, test policy, test suite, test summary report, testware

Background

The most visible part of testing is test execution. But to be effective and efficient, test plans should also include time to be spent on planning the tests, designing test cases, preparing for execution and evaluating results.

The fundamental test process consists of the following main activities:

- Test planning and control
- Test analysis and design
- Test implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

Although logically sequential, the activities in the process may overlap or take place concurrently. Tailoring these main activities within the context of the system and the project is usually required.

1.4.1 Test Planning and Control (K1)

Test planning is the activity of defining the objectives of testing and the specification of test activities in order to meet the objectives and mission.

Test control is the ongoing activity of comparing actual progress against the plan, and reporting the status, including deviations from the plan. It involves taking actions necessary to meet the mission and objectives of the project. In order to control testing, the testing activities should be monitored throughout the project. Test planning takes into account the feedback from monitoring and control activities.

Test planning and control tasks are defined in Chapter 5 of this syllabus.

1.4.2 Test Analysis and Design (K1)

Test analysis and design is the activity during which general testing objectives are transformed into tangible test conditions and test cases.

The test analysis and design activity has the following major tasks:

- Reviewing the test basis (such as requirements, software integrity level¹ (risk level), risk analysis reports, architecture, design, interface specifications)
- Evaluating testability of the test basis and test objects
- Identifying and prioritizing test conditions based on analysis of test items, the specification, behavior and structure of the software
- Designing and prioritizing high level test cases
- Identifying necessary test data to support the test conditions and test cases
- Designing the test environment setup and identifying any required infrastructure and tools
- Creating bi-directional traceability between test basis and test cases

¹ The degree to which software complies or must comply with a set of stakeholder-selected software and/or software-based system characteristics (e.g., software complexity, risk assessment, safety level, security level, desired performance, reliability, or cost) which are defined to reflect the importance of the software to its stakeholders.

1.4.3 Test Implementation and Execution (K1)

Test implementation and execution is the activity where test procedures or scripts are specified by combining the test cases in a particular order and including any other information needed for test execution, the environment is set up and the tests are run.

Test implementation and execution has the following major tasks:

- Finalizing, implementing and prioritizing test cases (including the identification of test data)
- Developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts
- Creating test suites from the test procedures for efficient test execution
- Verifying that the test environment has been set up correctly
- Verifying and updating bi-directional traceability between the test basis and test cases
- Executing test procedures either manually or by using test execution tools, according to the planned sequence
- Logging the outcome of test execution and recording the identities and versions of the software under test, test tools and testware
- Comparing actual results with expected results
- Reporting discrepancies as incidents and analyzing them in order to establish their cause (e.g., a defect in the code, in specified test data, in the test document, or a mistake in the way the test was executed)
- Repeating test activities as a result of action taken for each discrepancy, for example, re-execution of a test that previously failed in order to confirm a fix (confirmation testing), execution of a corrected test and/or execution of tests in order to ensure that defects have not been introduced in unchanged areas of the software or that defect fixing did not uncover other defects (regression testing)

1.4.4 Evaluating Exit Criteria and Reporting (K1)

Evaluating exit criteria is the activity where test execution is assessed against the defined objectives. This should be done for each test level (see Section 2.2).

Evaluating exit criteria has the following major tasks:

- Checking test logs against the exit criteria specified in test planning
- Assessing if more tests are needed or if the exit criteria specified should be changed
- Writing a test summary report for stakeholders

1.4.5 Test Closure Activities (K1)

Test closure activities collect data from completed test activities to consolidate experience, testware, facts and numbers. Test closure activities occur at project milestones such as when a software system is released, a test project is completed (or cancelled), a milestone has been achieved, or a maintenance release has been completed.

Test closure activities include the following major tasks:

- Checking which planned deliverables have been delivered
- Closing incident reports or raising change records for any that remain open
- Documenting the acceptance of the system
- Finalizing and archiving testware, the test environment and the test infrastructure for later reuse
- Handing over the testware to the maintenance organization
- Analyzing lessons learned to determine changes needed for future releases and projects
- Using the information gathered to improve test maturity

1.5 The Psychology of Testing (K2)

25 minutes

Terms

Error guessing, independence

Background

The mindset to be used while testing and reviewing is different from that used while developing software. With the right mindset developers are able to test their own code, but separation of this responsibility to a tester is typically done to help focus effort and provide additional benefits, such as an independent view by trained and professional testing resources. Independent testing may be carried out at any level of testing.

A certain degree of independence (avoiding the author bias) often makes the tester more effective at finding defects and failures. Independence is not, however, a replacement for familiarity, and developers can efficiently find many defects in their own code. Several levels of independence can be defined as shown here from low to high:

- Tests designed by the person(s) who wrote the software under test (low level of independence)
- Tests designed by another person(s) (e.g., from the development team)
- Tests designed by a person(s) from a different organizational group (e.g., an independent test team) or test specialists (e.g., usability or performance test specialists)
- Tests designed by a person(s) from a different organization or company (i.e., outsourcing or certification by an external body)

People and projects are driven by objectives. People tend to align their plans with the objectives set by management and other stakeholders, for example, to find defects or to confirm that software meets its objectives. Therefore, it is important to clearly state the objectives of testing.

Identifying failures during testing may be perceived as criticism against the product and against the author. As a result, testing is often seen as a destructive activity, even though it is very constructive in the management of product risks. Looking for failures in a system requires curiosity, professional pessimism, a critical eye, attention to detail, good communication with development peers, and experience on which to base error guessing.

If errors, defects or failures are communicated in a constructive way, bad feelings between the testers and the analysts, designers and developers can be avoided. This applies to defects found during reviews as well as in testing.

The tester and test leader need good interpersonal skills to communicate factual information about defects, progress and risks in a constructive way. For the author of the software or document, defect information can help them improve their skills. Defects found and fixed during testing will save time and money later, and reduce risks.

Communication problems may occur, particularly if testers are seen only as messengers of unwanted news about defects. However, there are several ways to improve communication and relationships between testers and others:

- Start with collaboration rather than battles – remind everyone of the common goal of better quality systems
- Communicate findings on the product in a neutral, fact-focused way without criticizing the person who created it, for example, write objective and factual incident reports and review findings
- Try to understand how the other person feels and why they react as they do
- Confirm that the other person has understood what you have said and vice versa

1.6 Code of Ethics	10 minutes
--------------------	------------

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use. Recognizing the ACM and IEEE code of ethics for engineers, the ISTQB states the following code of ethics:

PUBLIC - Certified software testers shall act consistently with the public interest

CLIENT AND EMPLOYER - Certified software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest

PRODUCT - Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible

JUDGMENT - Certified software testers shall maintain integrity and independence in their professional judgment

MANAGEMENT - Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing

PROFESSION - Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest

COLLEAGUES - Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers

SELF - Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession

References

- 1.1.5 Black, 2001, Kaner, 2002
- 1.2 Beizer, 1990, Black, 2001, Myers, 1979
- 1.3 Beizer, 1990, Hetzel, 1988, Myers, 1979
- 1.4 Hetzel, 1988
- 1.4.5 Black, 2001, Craig, 2002
- 1.5 Black, 2001, Hetzel, 1988

2. Testing Throughout the Software Life Cycle (K2)	115 minutes
-----------------------------------------------------------	--------------------

Learning Objectives for Testing Throughout the Software Life Cycle

The objectives identify what you will be able to do following the completion of each module.

2.1 Software Development Models (K2)

- LO-2.1.1 Explain the relationship between development, test activities and work products in the development life cycle, by giving examples using project and product types (K2)
- LO-2.1.2 Recognize the fact that software development models must be adapted to the context of project and product characteristics (K1)
- LO-2.1.3 Recall characteristics of good testing that are applicable to any life cycle model (K1)

2.2 Test Levels (K2)

- LO-2.2.1 Compare the different levels of testing: major objectives, typical objects of testing, typical targets of testing (e.g., functional or structural) and related work products, people who test, types of defects and failures to be identified (K2)

2.3 Test Types (K2)

- LO-2.3.1 Compare four software test types (functional, non-functional, structural and change-related) by example (K2)
- LO-2.3.2 Recognize that functional and structural tests occur at any test level (K1)
- LO-2.3.3 Identify and describe non-functional test types based on non-functional requirements (K2)
- LO-2.3.4 Identify and describe test types based on the analysis of a software system's structure or architecture (K2)
- LO-2.3.5 Describe the purpose of confirmation testing and regression testing (K2)

2.4 Maintenance Testing (K2)

- LO-2.4.1 Compare maintenance testing (testing an existing system) to testing a new application with respect to test types, triggers for testing and amount of testing (K2)
- LO-2.4.2 Recognize indicators for maintenance testing (modification, migration and retirement) (K1)
- LO-2.4.3 Describe the role of regression testing and impact analysis in maintenance (K2)

2.1 Software Development Models (K2)

20 minutes

Terms

Commercial Off-The-Shelf (COTS), iterative-incremental development model, validation, verification, V-model

Background

Testing does not exist in isolation; test activities are related to software development activities. Different development life cycle models need different approaches to testing.

2.1.1 V-model (Sequential Development Model) (K2)

Although variants of the V-model exist, a common type of V-model uses four test levels, corresponding to the four development levels.

The four levels used in this syllabus are:

- Component (unit) testing
- Integration testing
- System testing
- Acceptance testing

In practice, a V-model may have more, fewer or different levels of development and testing, depending on the project and the software product. For example, there may be component integration testing after component testing, and system integration testing after system testing.

Software work products (such as business scenarios or use cases, requirements specifications, design documents and code) produced during development are often the basis of testing in one or more test levels. References for generic work products include Capability Maturity Model Integration (CMMI) or 'Software life cycle processes' (IEEE/IEC 12207). Verification and validation (and early test design) can be carried out during the development of the software work products.

2.1.2 Iterative-incremental Development Models (K2)

Iterative-incremental development is the process of establishing requirements, designing, building and testing a system in a series of short development cycles. Examples are: prototyping, Rapid Application Development (RAD), Rational Unified Process (RUP) and agile development models. A system that is produced using these models may be tested at several test levels during each iteration. An increment, added to others developed previously, forms a growing partial system, which should also be tested. Regression testing is increasingly important on all iterations after the first one. Verification and validation can be carried out on each increment.

2.1.3 Testing within a Life Cycle Model (K2)

In any life cycle model, there are several characteristics of good testing:

- For every development activity there is a corresponding testing activity
- Each test level has test objectives specific to that level
- The analysis and design of tests for a given test level should begin during the corresponding development activity
- Testers should be involved in reviewing documents as soon as drafts are available in the development life cycle

Test levels can be combined or reorganized depending on the nature of the project or the system architecture. For example, for the integration of a Commercial Off-The-Shelf (COTS) software product into a system, the purchaser may perform integration testing at the system level (e.g.,

integration to the infrastructure and other systems, or system deployment) and acceptance testing (functional and/or non-functional, and user and/or operational testing).

2.2 Test Levels (K2)

40 minutes

Terms

Alpha testing, beta testing, component testing, driver, field testing, functional requirement, integration, integration testing, non-functional requirement, robustness testing, stub, system testing, test environment, test level, test-driven development, user acceptance testing

Background

For each of the test levels, the following can be identified: the generic objectives, the work product(s) being referenced for deriving test cases (i.e., the test basis), the test object (i.e., what is being tested), typical defects and failures to be found, test harness requirements and tool support, and specific approaches and responsibilities.

Testing a system's configuration data shall be considered during test planning,

2.2.1 Component Testing (K2)

Test basis:

- Component requirements
- Detailed design
- Code

Typical test objects:

- Components
- Programs
- Data conversion / migration programs
- Database modules

Component testing (also known as unit, module or program testing) searches for defects in, and verifies the functioning of, software modules, programs, objects, classes, etc., that are separately testable. It may be done in isolation from the rest of the system, depending on the context of the development life cycle and the system. Stubs, drivers and simulators may be used.

Component testing may include testing of functionality and specific non-functional characteristics, such as resource-behavior (e.g., searching for memory leaks) or robustness testing, as well as structural testing (e.g., decision coverage). Test cases are derived from work products such as a specification of the component, the software design or the data model.

Typically, component testing occurs with access to the code being tested and with the support of a development environment, such as a unit test framework or debugging tool. In practice, component testing usually involves the programmer who wrote the code. Defects are typically fixed as soon as they are found, without formally managing these defects.

One approach to component testing is to prepare and automate test cases before coding. This is called a test-first approach or test-driven development. This approach is highly iterative and is based on cycles of developing test cases, then building and integrating small pieces of code, and executing the component tests correcting any issues and iterating until they pass.

2.2.2 Integration Testing (K2)

Test basis:

- Software and system design
- Architecture
- Workflows
- Use cases

Typical test objects:

- Subsystems
- Database implementation
- Infrastructure
- Interfaces
- System configuration and configuration data

Integration testing tests interfaces between components, interactions with different parts of a system, such as the operating system, file system and hardware, and interfaces between systems.

There may be more than one level of integration testing and it may be carried out on test objects of varying size as follows:

1. Component integration testing tests the interactions between software components and is done after component testing
2. System integration testing tests the interactions between different systems or between hardware and software and may be done after system testing. In this case, the developing organization may control only one side of the interface. This might be considered as a risk. Business processes implemented as workflows may involve a series of systems. Cross-platform issues may be significant.

The greater the scope of integration, the more difficult it becomes to isolate defects to a specific component or system, which may lead to increased risk and additional time for troubleshooting.

Systematic integration strategies may be based on the system architecture (such as top-down and bottom-up), functional tasks, transaction processing sequences, or some other aspect of the system or components. In order to ease fault isolation and detect defects early, integration should normally be incremental rather than “big bang”.

Testing of specific non-functional characteristics (e.g., performance) may be included in integration testing as well as functional testing.

At each stage of integration, testers concentrate solely on the integration itself. For example, if they are integrating module A with module B they are interested in testing the communication between the modules, not the functionality of the individual module as that was done during component testing. Both functional and structural approaches may be used.

Ideally, testers should understand the architecture and influence integration planning. If integration tests are planned before components or systems are built, those components can be built in the order required for most efficient testing.

2.2.3 System Testing (K2)

Test basis:

- System and software requirement specification
- Use cases
- Functional specification
- Risk analysis reports

Typical test objects:

- System, user and operation manuals
- System configuration and configuration data

System testing is concerned with the behavior of a whole system/product. The testing scope shall be clearly addressed in the Master and/or Level Test Plan for that test level.

In system testing, the test environment should correspond to the final target or production environment as much as possible in order to minimize the risk of environment-specific failures not being found in testing.

System testing may include tests based on risks and/or on requirements specifications, business processes, use cases, or other high level text descriptions or models of system behavior, interactions with the operating system, and system resources.

System testing should investigate functional and non-functional requirements of the system, and data quality characteristics. Testers also need to deal with incomplete or undocumented requirements. System testing of functional requirements starts by using the most appropriate specification-based (black-box) techniques for the aspect of the system to be tested. For example, a decision table may be created for combinations of effects described in business rules. Structure-based techniques (white-box) may then be used to assess the thoroughness of the testing with respect to a structural element, such as menu structure or web page navigation (see Chapter 4).

An independent test team often carries out system testing.

2.2.4 Acceptance Testing (K2)

Test basis:

- User requirements
- System requirements
- Use cases
- Business processes
- Risk analysis reports

Typical test objects:

- Business processes on fully integrated system
- Operational and maintenance processes
- User procedures
- Forms
- Reports
- Configuration data

Acceptance testing is often the responsibility of the customers or users of a system; other stakeholders may be involved as well.

The goal in acceptance testing is to establish confidence in the system, parts of the system or specific non-functional characteristics of the system. Finding defects is not the main focus in acceptance testing. Acceptance testing may assess the system's readiness for deployment and

use, although it is not necessarily the final level of testing. For example, a large-scale system integration test may come after the acceptance test for a system.

Acceptance testing may occur at various times in the life cycle, for example:

- A COTS software product may be acceptance tested when it is installed or integrated
- Acceptance testing of the usability of a component may be done during component testing
- Acceptance testing of a new functional enhancement may come before system testing

Typical forms of acceptance testing include the following:

User acceptance testing

Typically verifies the fitness for use of the system by business users.

Operational (acceptance) testing

The acceptance of the system by the system administrators, including:

- Testing of backup/restore
- Disaster recovery
- User management
- Maintenance tasks
- Data load and migration tasks
- Periodic checks of security vulnerabilities

Contract and regulation acceptance testing

Contract acceptance testing is performed against a contract's acceptance criteria for producing custom-developed software. Acceptance criteria should be defined when the parties agree to the contract. Regulation acceptance testing is performed against any regulations that must be adhered to, such as government, legal or safety regulations.

Alpha and beta (or field) testing

Developers of market, or COTS, software often want to get feedback from potential or existing customers in their market before the software product is put up for sale commercially. Alpha testing is performed at the developing organization's site but not by the developing team. Beta testing, or field-testing, is performed by customers or potential customers at their own locations.

Organizations may use other terms as well, such as factory acceptance testing and site acceptance testing for systems that are tested before and after being moved to a customer's site.

2.3 Test Types (K2)

40 minutes

Terms

Black-box testing, code coverage, functional testing, interoperability testing, load testing, maintainability testing, performance testing, portability testing, reliability testing, security testing, stress testing, structural testing, usability testing, white-box testing

Background

A group of test activities can be aimed at verifying the software system (or a part of a system) based on a specific reason or target for testing.

A test type is focused on a particular test objective, which could be any of the following:

- A function to be performed by the software
- A non-functional quality characteristic, such as reliability or usability
- The structure or architecture of the software or system
- Change related, i.e., confirming that defects have been fixed (confirmation testing) and looking for unintended changes (regression testing)

A model of the software may be developed and/or used in structural testing (e.g., a control flow model or menu structure model), non-functional testing (e.g., performance model, usability model security threat modeling), and functional testing (e.g., a process flow model, a state transition model or a plain language specification).

2.3.1 Testing of Function (Functional Testing) (K2)

The functions that a system, subsystem or component are to perform may be described in work products such as a requirements specification, use cases, or a functional specification, or they may be undocumented. The functions are “what” the system does.

Functional tests are based on functions and features (described in documents or understood by the testers) and their interoperability with specific systems, and may be performed at all test levels (e.g., tests for components may be based on a component specification).

Specification-based techniques may be used to derive test conditions and test cases from the functionality of the software or system (see Chapter 4). Functional testing considers the external behavior of the software (black-box testing).

A type of functional testing, security testing, investigates the functions (e.g., a firewall) relating to detection of threats, such as viruses, from malicious outsiders. Another type of functional testing, interoperability testing, evaluates the capability of the software product to interact with one or more specified components or systems.

2.3.2 Testing of Non-functional Software Characteristics (Non-functional Testing) (K2)

Non-functional testing includes, but is not limited to, performance testing, load testing, stress testing, usability testing, maintainability testing, reliability testing and portability testing. It is the testing of “how” the system works.

Non-functional testing may be performed at all test levels. The term non-functional testing describes the tests required to measure characteristics of systems and software that can be quantified on a varying scale, such as response times for performance testing. These tests can be referenced to a quality model such as the one defined in ‘Software Engineering – Software Product Quality’ (ISO

9126). Non-functional testing considers the external behavior of the software and in most cases uses black-box test design techniques to accomplish that.

2.3.3 Testing of Software Structure/Architecture (Structural Testing) (K2)

Structural (white-box) testing may be performed at all test levels. Structural techniques are best used after specification-based techniques, in order to help measure the thoroughness of testing through assessment of coverage of a type of structure.

Coverage is the extent that a structure has been exercised by a test suite, expressed as a percentage of the items being covered. If coverage is not 100%, then more tests may be designed to test those items that were missed to increase coverage. Coverage techniques are covered in Chapter 4.

At all test levels, but especially in component testing and component integration testing, tools can be used to measure the code coverage of elements, such as statements or decisions. Structural testing may be based on the architecture of the system, such as a calling hierarchy.

Structural testing approaches can also be applied at system, system integration or acceptance testing levels (e.g., to business models or menu structures).

2.3.4 Testing Related to Changes: Re-testing and Regression Testing (K2)

After a defect is detected and fixed, the software should be re-tested to confirm that the original defect has been successfully removed. This is called confirmation. Debugging (locating and fixing a defect) is a development activity, not a testing activity.

Regression testing is the repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the change(s). These defects may be either in the software being tested, or in another related or unrelated software component. It is performed when the software, or its environment, is changed. The extent of regression testing is based on the risk of not finding defects in software that was working previously.

Tests should be repeatable if they are to be used for confirmation testing and to assist regression testing.

Regression testing may be performed at all test levels, and includes functional, non-functional and structural testing. Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation.

2.4 Maintenance Testing (K2)

15 minutes

Terms

Impact analysis, maintenance testing

Background

Once deployed, a software system is often in service for years or decades. During this time the system, its configuration data, or its environment are often corrected, changed or extended. The planning of releases in advance is crucial for successful maintenance testing. A distinction has to be made between planned releases and hot fixes. Maintenance testing is done on an existing operational system, and is triggered by modifications, migration, or retirement of the software or system.

Modifications include planned enhancement changes (e.g., release-based), corrective and emergency changes, and changes of environment, such as planned operating system or database upgrades, planned upgrade of Commercial-Off-The-Shelf software, or patches to correct newly exposed or discovered vulnerabilities of the operating system.

Maintenance testing for migration (e.g., from one platform to another) should include operational tests of the new environment as well as of the changed software. Migration testing (conversion testing) is also needed when data from another application will be migrated into the system being maintained.

Maintenance testing for the retirement of a system may include the testing of data migration or archiving if long data-retention periods are required.

In addition to testing what has been changed, maintenance testing includes regression testing to parts of the system that have not been changed. The scope of maintenance testing is related to the risk of the change, the size of the existing system and to the size of the change. Depending on the changes, maintenance testing may be done at any or all test levels and for any or all test types. Determining how the existing system may be affected by changes is called impact analysis, and is used to help decide how much regression testing to do. The impact analysis may be used to determine the regression test suite.

Maintenance testing can be difficult if specifications are out of date or missing, or testers with domain knowledge are not available.

References

- 2.1.3 CMMI, Craig, 2002, Hetzel, 1988, IEEE 12207
- 2.2 Hetzel, 1988
- 2.2.4 Copeland, 2004, Myers, 1979
- 2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
- 2.3.2 Black, 2001, ISO 9126
- 2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1988
- 2.3.4 Hetzel, 1988, IEEE STD 829-1998
- 2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE STD 829-1998

3. Static Techniques (K2)	60 minutes
----------------------------------	-------------------

Learning Objectives for Static Techniques

The objectives identify what you will be able to do following the completion of each module.

3.1 Static Techniques and the Test Process (K2)

- LO-3.1.1 Recognize software work products that can be examined by the different static techniques (K1)
- LO-3.1.2 Describe the importance and value of considering static techniques for the assessment of software work products (K2)
- LO-3.1.3 Explain the difference between static and dynamic techniques, considering objectives, types of defects to be identified, and the role of these techniques within the software life cycle (K2)

3.2 Review Process (K2)

- LO-3.2.1 Recall the activities, roles and responsibilities of a typical formal review (K1)
- LO-3.2.2 Explain the differences between different types of reviews: informal review, technical review, walkthrough and inspection (K2)
- LO-3.2.3 Explain the factors for successful performance of reviews (K2)

3.3 Static Analysis by Tools (K2)

- LO-3.3.1 Recall typical defects and errors identified by static analysis and compare them to reviews and dynamic testing (K1)
- LO-3.3.2 Describe, using examples, the typical benefits of static analysis (K2)
- LO-3.3.3 List typical code and design defects that may be identified by static analysis tools (K1)

3.1 Static Techniques and the Test Process (K2)

15 minutes

Terms

Dynamic testing, static testing

Background

Unlike dynamic testing, which requires the execution of software, static testing techniques rely on the manual examination (reviews) and automated analysis (static analysis) of the code or other project documentation without the execution of the code.

Reviews are a way of testing software work products (including code) and can be performed well before dynamic test execution. Defects detected during reviews early in the life cycle (e.g., defects found in requirements) are often much cheaper to remove than those detected by running tests on the executing code.

A review could be done entirely as a manual activity, but there is also tool support. The main manual activity is to examine a work product and make comments about it. Any software work product can be reviewed, including requirements specifications, design specifications, code, test plans, test specifications, test cases, test scripts, user guides or web pages.

Benefits of reviews include early defect detection and correction, development productivity improvements, reduced development timescales, reduced testing cost and time, lifetime cost reductions, fewer defects and improved communication. Reviews can find omissions, for example, in requirements, which are unlikely to be found in dynamic testing.

Reviews, static analysis and dynamic testing have the same objective – identifying defects. They are complementary; the different techniques can find different types of defects effectively and efficiently. Compared to dynamic testing, static techniques find causes of failures (defects) rather than the failures themselves.

Typical defects that are easier to find in reviews than in dynamic testing include: deviations from standards, requirement defects, design defects, insufficient maintainability and incorrect interface specifications.

3.2 Review Process (K2)	25 minutes
-------------------------	------------

Terms

Entry criteria, formal review, informal review, inspection, metric, moderator, peer review, reviewer, scribe, technical review, walkthrough

Background

The different types of reviews vary from informal, characterized by no written instructions for reviewers, to systematic, characterized by team participation, documented results of the review, and documented procedures for conducting the review. The formality of a review process is related to factors such as the maturity of the development process, any legal or regulatory requirements or the need for an audit trail.

The way a review is carried out depends on the agreed objectives of the review (e.g., find defects, gain understanding, educate testers and new team members, or discussion and decision by consensus).

3.2.1 Activities of a Formal Review (K1)

A typical formal review has the following main activities:

1. Planning
 - Defining the review criteria
 - Selecting the personnel
 - Allocating roles
 - Defining the entry and exit criteria for more formal review types (e.g., inspections)
 - Selecting which parts of documents to review
 - Checking entry criteria (for more formal review types)
2. Kick-off
 - Distributing documents
 - Explaining the objectives, process and documents to the participants
3. Individual preparation
 - Preparing for the review meeting by reviewing the document(s)
 - Noting potential defects, questions and comments
4. Examination/evaluation/recording of results (review meeting)
 - Discussing or logging, with documented results or minutes (for more formal review types)
 - Noting defects, making recommendations regarding handling the defects, making decisions about the defects
 - Examining/evaluating and recording issues during any physical meetings or tracking any group electronic communications
5. Rework
 - Fixing defects found (typically done by the author)
 - Recording updated status of defects (in formal reviews)
6. Follow-up
 - Checking that defects have been addressed
 - Gathering metrics
 - Checking on exit criteria (for more formal review types)

3.2.2 Roles and Responsibilities (K1)

A typical formal review will include the roles below:

- Manager: decides on the execution of reviews, allocates time in project schedules and determines if the review objectives have been met.

- Moderator: the person who leads the review of the document or set of documents, including planning the review, running the meeting, and following-up after the meeting. If necessary, the moderator may mediate between the various points of view and is often the person upon whom the success of the review rests.
- Author: the writer or person with chief responsibility for the document(s) to be reviewed.
- Reviewers: individuals with a specific technical or business background (also called checkers or inspectors) who, after the necessary preparation, identify and describe findings (e.g., defects) in the product under review. Reviewers should be chosen to represent different perspectives and roles in the review process, and should take part in any review meetings.
- Scribe (or recorder): documents all the issues, problems and open points that were identified during the meeting.

Looking at software products or related work products from different perspectives and using checklists can make reviews more effective and efficient. For example, a checklist based on various perspectives such as user, maintainer, tester or operations, or a checklist of typical requirements problems may help to uncover previously undetected issues.

3.2.3 Types of Reviews (K2)

A single software product or related work product may be the subject of more than one review. If more than one type of review is used, the order may vary. For example, an informal review may be carried out before a technical review, or an inspection may be carried out on a requirements specification before a walkthrough with customers. The main characteristics, options and purposes of common review types are:

Informal Review

- No formal process
- May take the form of pair programming or a technical lead reviewing designs and code
- Results may be documented
- Varies in usefulness depending on the reviewers
- Main purpose: inexpensive way to get some benefit

Walkthrough

- Meeting led by author
- May take the form of scenarios, dry runs, peer group participation
- Open-ended sessions
 - Optional pre-meeting preparation of reviewers
 - Optional preparation of a review report including list of findings
- Optional scribe (who is not the author)
- May vary in practice from quite informal to very formal
- Main purposes: learning, gaining understanding, finding defects

Technical Review

- Documented, defined defect-detection process that includes peers and technical experts with optional management participation
- May be performed as a peer review without management participation
- Ideally led by trained moderator (not the author)
- Pre-meeting preparation by reviewers
- Optional use of checklists
- Preparation of a review report which includes the list of findings, the verdict whether the software product meets its requirements and, where appropriate, recommendations related to findings
- May vary in practice from quite informal to very formal
- Main purposes: discussing, making decisions, evaluating alternatives, finding defects, solving technical problems and checking conformance to specifications, plans, regulations, and standards

Inspection

- Led by trained moderator (not the author)
- Usually conducted as a peer examination
- Defined roles
- Includes metrics gathering
- Formal process based on rules and checklists
- Specified entry and exit criteria for acceptance of the software product
- Pre-meeting preparation
- Inspection report including list of findings
- Formal follow-up process (with optional process improvement components)
- Optional reader
- Main purpose: finding defects

Walkthroughs, technical reviews and inspections can be performed within a peer group, i.e., colleagues at the same organizational level. This type of review is called a “peer review”.

3.2.4 Success Factors for Reviews (K2)

Success factors for reviews include:

- Each review has clear predefined objectives
- The right people for the review objectives are involved
- Testers are valued reviewers who contribute to the review and also learn about the product which enables them to prepare tests earlier
- Defects found are welcomed and expressed objectively
- People issues and psychological aspects are dealt with (e.g., making it a positive experience for the author)
- The review is conducted in an atmosphere of trust; the outcome will not be used for the evaluation of the participants
- Review techniques are applied that are suitable to achieve the objectives and to the type and level of software work products and reviewers
- Checklists or roles are used if appropriate to increase effectiveness of defect identification
- Training is given in review techniques, especially the more formal techniques such as inspection
- Management supports a good review process (e.g., by incorporating adequate time for review activities in project schedules)
- There is an emphasis on learning and process improvement

3.3 Static Analysis by Tools (K2)

20 minutes

Terms

Compiler, complexity, control flow, data flow, static analysis

Background

The objective of static analysis is to find defects in software source code and software models. Static analysis is performed without actually executing the software being examined by the tool; dynamic testing does execute the software code. Static analysis can locate defects that are hard to find in dynamic testing. As with reviews, static analysis finds defects rather than failures. Static analysis tools analyze program code (e.g., control flow and data flow), as well as generated output such as HTML and XML.

The value of static analysis is:

- Early detection of defects prior to test execution
- Early warning about suspicious aspects of the code or design by the calculation of metrics, such as a high complexity measure
- Identification of defects not easily found by dynamic testing
- Detecting dependencies and inconsistencies in software models such as links
- Improved maintainability of code and design
- Prevention of defects, if lessons are learned in development

Typical defects discovered by static analysis tools include:

- Referencing a variable with an undefined value
- Inconsistent interfaces between modules and components
- Variables that are not used or are improperly declared
- Unreachable (dead) code
- Missing and erroneous logic (potentially infinite loops)
- Overly complicated constructs
- Programming standards violations
- Security vulnerabilities
- Syntax violations of code and software models

Static analysis tools are typically used by developers (checking against predefined rules or programming standards) before and during component and integration testing or when checking-in code to configuration management tools, and by designers during software modeling. Static analysis tools may produce a large number of warning messages, which need to be well-managed to allow the most effective use of the tool.

Compilers may offer some support for static analysis, including the calculation of metrics.

References

3.2 IEEE 1028

3.2.2 Gilb, 1993, van Veenendaal, 2004

3.2.4 Gilb, 1993, IEEE 1028

3.3 van Veenendaal, 2004

4. Test Design Techniques (K4)	285 minutes
---------------------------------------	--------------------

Learning Objectives for Test Design Techniques

The objectives identify what you will be able to do following the completion of each module.

4.1 The Test Development Process (K3)

- LO-4.1.1 Differentiate between a test design specification, test case specification and test procedure specification (K2)
- LO-4.1.2 Compare the terms test condition, test case and test procedure (K2)
- LO-4.1.3 Evaluate the quality of test cases in terms of clear traceability to the requirements and expected results (K2)
- LO-4.1.4 Translate test cases into a well-structured test procedure specification at a level of detail relevant to the knowledge of the testers (K3)

4.2 Categories of Test Design Techniques (K2)

- LO-4.2.1 Recall reasons that both specification-based (black-box) and structure-based (white-box) test design techniques are useful and list the common techniques for each (K1)
- LO-4.2.2 Explain the characteristics, commonalities, and differences between specification-based testing, structure-based testing and experience-based testing (K2)

4.3 Specification-based or Black-box Techniques (K3)

- LO-4.3.1 Write test cases from given software models using equivalence partitioning, boundary value analysis, decision tables and state transition diagrams/tables (K3)
- LO-4.3.2 Explain the main purpose of each of the four testing techniques, what level and type of testing could use the technique, and how coverage may be measured (K2)
- LO-4.3.3 Explain the concept of use case testing and its benefits (K2)

4.4 Structure-based or White-box Techniques (K4)

- LO-4.4.1 Describe the concept and value of code coverage (K2)
- LO-4.4.2 Explain the concepts of statement and decision coverage, and give reasons why these concepts can also be used at test levels other than component testing (e.g., on business procedures at system level) (K2)
- LO-4.4.3 Write test cases from given control flows using statement and decision test design techniques (K3)
- LO-4.4.4 Assess statement and decision coverage for completeness with respect to defined exit criteria. (K4)

4.5 Experience-based Techniques (K2)

- LO-4.5.1 Recall reasons for writing test cases based on intuition, experience and knowledge about common defects (K1)
- LO-4.5.2 Compare experience-based techniques with specification-based testing techniques (K2)

4.6 Choosing Test Techniques (K2)

- LO-4.6.1 Classify test design techniques according to their fitness to a given context, for the test basis, respective models and software characteristics (K2)

4.1 The Test Development Process (K3)

15 minutes

Terms

Test case specification, test design, test execution schedule, test procedure specification, test script, traceability

Background

The test development process described in this section can be done in different ways, from very informal with little or no documentation, to very formal (as it is described below). The level of formality depends on the context of the testing, including the maturity of testing and development processes, time constraints, safety or regulatory requirements, and the people involved.

During test analysis, the test basis documentation is analyzed in order to determine what to test, i.e., to identify the test conditions. A test condition is defined as an item or event that could be verified by one or more test cases (e.g., a function, transaction, quality characteristic or structural element).

Establishing traceability from test conditions back to the specifications and requirements enables both effective impact analysis when requirements change, and determining requirements coverage for a set of tests. During test analysis the detailed test approach is implemented to select the test design techniques to use based on, among other considerations, the identified risks (see Chapter 5 for more on risk analysis).

During test design the test cases and test data are created and specified. A test case consists of a set of input values, execution preconditions, expected results and execution postconditions, defined to cover a certain test objective(s) or test condition(s). The 'Standard for Software Test Documentation' (IEEE STD 829-1998) describes the content of test design specifications (containing test conditions) and test case specifications.

Expected results should be produced as part of the specification of a test case and include outputs, changes to data and states, and any other consequences of the test. If expected results have not been defined, then a plausible, but erroneous, result may be interpreted as the correct one. Expected results should ideally be defined prior to test execution.

During test implementation the test cases are developed, implemented, prioritized and organized in the test procedure specification (IEEE STD 829-1998). The test procedure specifies the sequence of actions for the execution of a test. If tests are run using a test execution tool, the sequence of actions is specified in a test script (which is an automated test procedure).

The various test procedures and automated test scripts are subsequently formed into a test execution schedule that defines the order in which the various test procedures, and possibly automated test scripts, are executed. The test execution schedule will take into account such factors as regression tests, prioritization, and technical and logical dependencies.

4.2 Categories of Test Design Techniques (K2)

15 minutes

Terms

Black-box test design technique, experience-based test design technique, test design technique, white-box test design technique

Background

The purpose of a test design technique is to identify test conditions, test cases, and test data.

It is a classic distinction to denote test techniques as black-box or white-box. Black-box test design techniques (also called specification-based techniques) are a way to derive and select test conditions, test cases, or test data based on an analysis of the test basis documentation. This includes both functional and non-functional testing. Black-box testing, by definition, does not use any information regarding the internal structure of the component or system to be tested. White-box test design techniques (also called structural or structure-based techniques) are based on an analysis of the structure of the component or system. Black-box and white-box testing may also be combined with experience-based techniques to leverage the experience of developers, testers and users to determine what should be tested.

Some techniques fall clearly into a single category; others have elements of more than one category.

This syllabus refers to specification-based test design techniques as black-box techniques and structure-based test design techniques as white-box techniques. In addition experience-based test design techniques are covered.

Common characteristics of specification-based test design techniques include:

- Models, either formal or informal, are used for the specification of the problem to be solved, the software or its components
- Test cases can be derived systematically from these models

Common characteristics of structure-based test design techniques include:

- Information about how the software is constructed is used to derive the test cases (e.g., code and detailed design information)
- The extent of coverage of the software can be measured for existing test cases, and further test cases can be derived systematically to increase coverage

Common characteristics of experience-based test design techniques include:

- The knowledge and experience of people are used to derive the test cases
- The knowledge of testers, developers, users and other stakeholders about the software, its usage and its environment is one source of information
- Knowledge about likely defects and their distribution is another source of information

4.3 Specification-based or Black-box Techniques (K3)

150 minutes

Terms

Boundary value analysis, decision table testing, equivalence partitioning, state transition testing, use case testing

4.3.1 Equivalence Partitioning (K3)

In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be processed in the same way. Equivalence partitions (or classes) can be found for both valid data, i.e., values that should be accepted and invalid data, i.e., values that should be rejected. Partitions can also be identified for outputs, internal values, time-related values (e.g., before or after an event) and for interface parameters (e.g., integrated components being tested during integration testing). Tests can be designed to cover all valid and invalid partitions. Equivalence partitioning is applicable at all levels of testing.

Equivalence partitioning can be used to achieve input and output coverage goals. It can be applied to human input, input via interfaces to a system, or interface parameters in integration testing.

4.3.2 Boundary Value Analysis (K3)

Behavior at the edge of each equivalence partition is more likely to be incorrect than behavior within the partition, so boundaries are an area where testing is likely to yield defects. The maximum and minimum values of a partition are its boundary values. A boundary value for a valid partition is a valid boundary value; the boundary of an invalid partition is an invalid boundary value. Tests can be designed to cover both valid and invalid boundary values. When designing test cases, a test for each boundary value is chosen.

Boundary value analysis can be applied at all test levels. It is relatively easy to apply and its defect-finding capability is high. Detailed specifications are helpful in determining the interesting boundaries.

This technique is often considered as an extension of equivalence partitioning or other black-box test design techniques. It can be used on equivalence classes for user input on screen as well as, for example, on time ranges (e.g., time out, transactional speed requirements) or table ranges (e.g., table size is 256*256).

4.3.3 Decision Table Testing (K3)

Decision tables are a good way to capture system requirements that contain logical conditions, and to document internal system design. They may be used to record complex business rules that a system is to implement. When creating decision tables, the specification is analyzed, and conditions and actions of the system are identified. The input conditions and actions are most often stated in such a way that they must be true or false (Boolean). The decision table contains the triggering conditions, often combinations of true and false for all input conditions, and the resulting actions for each combination of conditions. Each column of the table corresponds to a business rule that defines a unique combination of conditions and which result in the execution of the actions associated with that rule. The coverage standard commonly used with decision table testing is to have at least one test per column in the table, which typically involves covering all combinations of triggering conditions.

The strength of decision table testing is that it creates combinations of conditions that otherwise might not have been exercised during testing. It may be applied to all situations when the action of the software depends on several logical decisions.

4.3.4 State Transition Testing (K3)

A system may exhibit a different response depending on current conditions or previous history (its state). In this case, that aspect of the system can be shown with a state transition diagram. It allows the tester to view the software in terms of its states, transitions between states, the inputs or events that trigger state changes (transitions) and the actions which may result from those transitions. The states of the system or object under test are separate, identifiable and finite in number.

A state table shows the relationship between the states and inputs, and can highlight possible transitions that are invalid.

Tests can be designed to cover a typical sequence of states, to cover every state, to exercise every transition, to exercise specific sequences of transitions or to test invalid transitions.

State transition testing is much used within the embedded software industry and technical automation in general. However, the technique is also suitable for modeling a business object having specific states or testing screen-dialogue flows (e.g., for Internet applications or business scenarios).

4.3.5 Use Case Testing (K2)

Tests can be derived from use cases. A use case describes interactions between actors (users or systems), which produce a result of value to a system user or the customer. Use cases may be described at the abstract level (business use case, technology-free, business process level) or at the system level (system use case on the system functionality level). Each use case has preconditions which need to be met for the use case to work successfully. Each use case terminates with postconditions which are the observable results and final state of the system after the use case has been completed. A use case usually has a mainstream (i.e., most likely) scenario and alternative scenarios.

Use cases describe the “process flows” through a system based on its actual likely use, so the test cases derived from use cases are most useful in uncovering defects in the process flows during real-world use of the system. Use cases are very useful for designing acceptance tests with customer/user participation. They also help uncover integration defects caused by the interaction and interference of different components, which individual component testing would not see. Designing test cases from use cases may be combined with other specification-based test techniques.

4.4 Structure-based or White-box Techniques (K4)

60 minutes

Terms

Code coverage, decision coverage, statement coverage, structure-based testing

Background

Structure-based or white-box testing is based on an identified structure of the software or the system, as seen in the following examples:

- Component level: the structure of a software component, i.e., statements, decisions, branches or even distinct paths
- Integration level: the structure may be a call tree (a diagram in which modules call other modules)
- System level: the structure may be a menu structure, business process or web page structure

In this section, three code-related structural test design techniques for code coverage, based on statements, branches and decisions, are discussed. For decision testing, a control flow diagram may be used to visualize the alternatives for each decision.

4.4.1 Statement Testing and Coverage (K4)

In component testing, statement coverage is the assessment of the percentage of executable statements that have been exercised by a test case suite. The statement testing technique derives test cases to execute specific statements, normally to increase statement coverage.

Statement coverage is determined by the number of executable statements covered by (designed or executed) test cases divided by the number of all executable statements in the code under test.

4.4.2 Decision Testing and Coverage (K4)

Decision coverage, related to branch testing, is the assessment of the percentage of decision outcomes (e.g., the True and False options of an IF statement) that have been exercised by a test case suite. The decision testing technique derives test cases to execute specific decision outcomes. Branches originate from decision points in the code and show the transfer of control to different locations in the code.

Decision coverage is determined by the number of all decision outcomes covered by (designed or executed) test cases divided by the number of all possible decision outcomes in the code under test.

Decision testing is a form of control flow testing as it follows a specific flow of control through the decision points. Decision coverage is stronger than statement coverage; 100% decision coverage guarantees 100% statement coverage, but not vice versa.

4.4.3 Other Structure-based Techniques (K1)

There are stronger levels of structural coverage beyond decision coverage, for example, condition coverage and multiple condition coverage.

The concept of coverage can also be applied at other test levels. For example, at the integration level the percentage of modules, components or classes that have been exercised by a test case suite could be expressed as module, component or class coverage.

Tool support is useful for the structural testing of code.

4.5 Experience-based Techniques (K2)	30 minutes
--------------------------------------	------------

Terms

Exploratory testing, (fault) attack

Background

Experience-based testing is where tests are derived from the tester's skill and intuition and their experience with similar applications and technologies. When used to augment systematic techniques, these techniques can be useful in identifying special tests not easily captured by formal techniques, especially when applied after more formal approaches. However, this technique may yield widely varying degrees of effectiveness, depending on the testers' experience.

A commonly used experience-based technique is error guessing. Generally testers anticipate defects based on experience. A structured approach to the error guessing technique is to enumerate a list of possible defects and to design tests that attack these defects. This systematic approach is called fault attack. These defect and failure lists can be built based on experience, available defect and failure data, and from common knowledge about why software fails.

Exploratory testing is concurrent test design, test execution, test logging and learning, based on a test charter containing test objectives, and carried out within time-boxes. It is an approach that is most useful where there are few or inadequate specifications and severe time pressure, or in order to augment or complement other, more formal testing. It can serve as a check on the test process, to help ensure that the most serious defects are found.

4.6 Choosing Test Techniques (K2)

15 minutes

Terms

No specific terms.

Background

The choice of which test techniques to use depends on a number of factors, including the type of system, regulatory standards, customer or contractual requirements, level of risk, type of risk, test objective, documentation available, knowledge of the testers, time and budget, development life cycle, use case models and previous experience with types of defects found.

Some techniques are more applicable to certain situations and test levels; others are applicable to all test levels.

When creating test cases, testers generally use a combination of test techniques including process, rule and data-driven techniques to ensure adequate coverage of the object under test.

References

- 4.1 Craig, 2002, Hetzel, 1988, IEEE STD 829-1998
- 4.2 Beizer, 1990, Copeland, 2004
- 4.3.1 Copeland, 2004, Myers, 1979
- 4.3.2 Copeland, 2004, Myers, 1979
- 4.3.3 Beizer, 1990, Copeland, 2004
- 4.3.4 Beizer, 1990, Copeland, 2004
- 4.3.5 Copeland, 2004
- 4.4.3 Beizer, 1990, Copeland, 2004
- 4.5 Kaner, 2002
- 4.6 Beizer, 1990, Copeland, 2004

5. Test Management (K3)

170 minutes

Learning Objectives for Test Management

The objectives identify what you will be able to do following the completion of each module.

5.1 Test Organization (K2)

- LO-5.1.1 Recognize the importance of independent testing (K1)
- LO-5.1.2 Explain the benefits and drawbacks of independent testing within an organization (K2)
- LO-5.1.3 Recognize the different team members to be considered for the creation of a test team (K1)
- LO-5.1.4 Recall the tasks of a typical test leader and tester (K1)

5.2 Test Planning and Estimation (K3)

- LO-5.2.1 Recognize the different levels and objectives of test planning (K1)
- LO-5.2.2 Summarize the purpose and content of the test plan, test design specification and test procedure documents according to the 'Standard for Software Test Documentation' (IEEE Std 829-1998) (K2)
- LO-5.2.3 Differentiate between conceptually different test approaches, such as analytical, model-based, methodical, process/standard compliant, dynamic/heuristic, consultative and regression-averse (K2)
- LO-5.2.4 Differentiate between the subject of test planning for a system and scheduling test execution (K2)
- LO-5.2.5 Write a test execution schedule for a given set of test cases, considering prioritization, and technical and logical dependencies (K3)
- LO-5.2.6 List test preparation and execution activities that should be considered during test planning (K1)
- LO-5.2.7 Recall typical factors that influence the effort related to testing (K1)
- LO-5.2.8 Differentiate between two conceptually different estimation approaches: the metrics-based approach and the expert-based approach (K2)
- LO-5.2.9 Recognize/justify adequate entry and exit criteria for specific test levels and groups of test cases (e.g., for integration testing, acceptance testing or test cases for usability testing) (K2)

5.3 Test Progress Monitoring and Control (K2)

- LO-5.3.1 Recall common metrics used for monitoring test preparation and execution (K1)
- LO-5.3.2 Explain and compare test metrics for test reporting and test control (e.g., defects found and fixed, and tests passed and failed) related to purpose and use (K2)
- LO-5.3.3 Summarize the purpose and content of the test summary report document according to the 'Standard for Software Test Documentation' (IEEE Std 829-1998) (K2)

5.4 Configuration Management (K2)

- LO-5.4.1 Summarize how configuration management supports testing (K2)

5.5 Risk and Testing (K2)

- LO-5.5.1 Describe a risk as a possible problem that would threaten the achievement of one or more stakeholders' project objectives (K2)
- LO-5.5.2 Remember that the level of risk is determined by likelihood (of happening) and impact (harm resulting if it does happen) (K1)
- LO-5.5.3 Distinguish between the project and product risks (K2)
- LO-5.5.4 Recognize typical product and project risks (K1)
- LO-5.5.5 Describe, using examples, how risk analysis and risk management may be used for test

5.6 Incident Management (K3)

- LO-5.6.1 Recognize the content of an incident report according to the 'Standard for Software Test Documentation' (IEEE Std 829-1998) (K1)
- LO-5.6.2 Write an incident report covering the observation of a failure during testing. (K3)

5.1 Test Organization (K2)	30 minutes
-----------------------------------	-------------------

Terms

Tester, test leader, test manager

5.1.1 Test Organization and Independence (K2)

The effectiveness of finding defects by testing and reviews can be improved by using independent testers. Options for independence include the following:

- No independent testers; developers test their own code
- Independent testers within the development teams
- Independent test team or group within the organization, reporting to project management or executive management
- Independent testers from the business organization or user community
- Independent test specialists for specific test types such as usability testers, security testers or certification testers (who certify a software product against standards and regulations)
- Independent testers outsourced or external to the organization

For large, complex or safety critical projects, it is usually best to have multiple levels of testing, with some or all of the levels done by independent testers. Development staff may participate in testing, especially at the lower levels, but their lack of objectivity often limits their effectiveness. The independent testers may have the authority to require and define test processes and rules, but testers should take on such process-related roles only in the presence of a clear management mandate to do so.

The benefits of independence include:

- Independent testers see other and different defects, and are unbiased
- An independent tester can verify assumptions people made during specification and implementation of the system

Drawbacks include:

- Isolation from the development team (if treated as totally independent)
- Developers may lose a sense of responsibility for quality
- Independent testers may be seen as a bottleneck or blamed for delays in release

Testing tasks may be done by people in a specific testing role, or may be done by someone in another role, such as a project manager, quality manager, developer, business and domain expert, infrastructure or IT operations.

5.1.2 Tasks of the Test Leader and Tester (K1)

In this syllabus two test positions are covered, test leader and tester. The activities and tasks performed by people in these two roles depend on the project and product context, the people in the roles, and the organization.

Sometimes the test leader is called a test manager or test coordinator. The role of the test leader may be performed by a project manager, a development manager, a quality assurance manager or the manager of a test group. In larger projects two positions may exist: test leader and test manager. Typically the test leader plans, monitors and controls the testing activities and tasks as defined in Section 1.4.

Typical test leader tasks may include:

- Coordinate the test strategy and plan with project managers and others
- Write or review a test strategy for the project, and test policy for the organization

- Contribute the testing perspective to other project activities, such as integration planning
- Plan the tests – considering the context and understanding the test objectives and risks – including selecting test approaches, estimating the time, effort and cost of testing, acquiring resources, defining test levels, cycles, and planning incident management
- Initiate the specification, preparation, implementation and execution of tests, monitor the test results and check the exit criteria
- Adapt planning based on test results and progress (sometimes documented in status reports) and take any action necessary to compensate for problems
- Set up adequate configuration management of testware for traceability
- Introduce suitable metrics for measuring test progress and evaluating the quality of the testing and the product
- Decide what should be automated, to what degree, and how
- Select tools to support testing and organize any training in tool use for testers
- Decide about the implementation of the test environment
- Write test summary reports based on the information gathered during testing

Typical tester tasks may include:

- Review and contribute to test plans
- Analyze, review and assess user requirements, specifications and models for testability
- Create test specifications
- Set up the test environment (often coordinating with system administration and network management)
- Prepare and acquire test data
- Implement tests on all test levels, execute and log the tests, evaluate the results and document the deviations from expected results
- Use test administration or management tools and test monitoring tools as required
- Automate tests (may be supported by a developer or a test automation expert)
- Measure performance of components and systems (if applicable)
- Review tests developed by others

People who work on test analysis, test design, specific test types or test automation may be specialists in these roles. Depending on the test level and the risks related to the product and the project, different people may take over the role of tester, keeping some degree of independence. Typically testers at the component and integration level would be developers, testers at the acceptance test level would be business experts and users, and testers for operational acceptance testing would be operators.

5.2 Test Planning and Estimation (K3)

40 minutes

Terms

Test approach, test strategy

5.2.1 Test Planning (K2)

This section covers the purpose of test planning within development and implementation projects, and for maintenance activities. Planning may be documented in a master test plan and in separate test plans for test levels such as system testing and acceptance testing. The outline of a test-planning document is covered by the 'Standard for Software Test Documentation' (IEEE Std 829-1998).

Planning is influenced by the test policy of the organization, the scope of testing, objectives, risks, constraints, criticality, testability and the availability of resources. As the project and test planning progress, more information becomes available and more detail can be included in the plan.

Test planning is a continuous activity and is performed in all life cycle processes and activities. Feedback from test activities is used to recognize changing risks so that planning can be adjusted.

5.2.2 Test Planning Activities (K3)

Test planning activities for an entire system or part of a system may include:

- Determining the scope and risks and identifying the objectives of testing
- Defining the overall approach of testing, including the definition of the test levels and entry and exit criteria
- Integrating and coordinating the testing activities into the software life cycle activities (acquisition, supply, development, operation and maintenance)
- Making decisions about what to test, what roles will perform the test activities, how the test activities should be done, and how the test results will be evaluated
- Scheduling test analysis and design activities
- Scheduling test implementation, execution and evaluation
- Assigning resources for the different activities defined
- Defining the amount, level of detail, structure and templates for the test documentation
- Selecting metrics for monitoring and controlling test preparation and execution, defect resolution and risk issues
- Setting the level of detail for test procedures in order to provide enough information to support reproducible test preparation and execution

5.2.3 Entry Criteria (K2)

Entry criteria define when to start testing such as at the beginning of a test level or when a set of tests is ready for execution.

Typically entry criteria may cover the following:

- Test environment availability and readiness
- Test tool readiness in the test environment
- Testable code availability
- Test data availability

5.2.4 Exit Criteria (K2)

Exit criteria define when to stop testing such as at the end of a test level or when a set of tests has achieved specific goal.

Typically exit criteria may cover the following:

- Thoroughness measures, such as coverage of code, functionality or risk
- Estimates of defect density or reliability measures
- Cost
- Residual risks, such as defects not fixed or lack of test coverage in certain areas
- Schedules such as those based on time to market

5.2.5 Test Estimation (K2)

Two approaches for the estimation of test effort are:

- The metrics-based approach: estimating the testing effort based on metrics of former or similar projects or based on typical values
- The expert-based approach: estimating the tasks based on estimates made by the owner of the tasks or by experts

Once the test effort is estimated, resources can be identified and a schedule can be drawn up.

The testing effort may depend on a number of factors, including:

- Characteristics of the product: the quality of the specification and other information used for test models (i.e., the test basis), the size of the product, the complexity of the problem domain, the requirements for reliability and security, and the requirements for documentation
- Characteristics of the development process: the stability of the organization, tools used, test process, skills of the people involved, and time pressure
- The outcome of testing: the number of defects and the amount of rework required

5.2.6 Test Strategy, Test Approach (K2)

The test approach is the implementation of the test strategy for a specific project. The test approach is defined and refined in the test plans and test designs. It typically includes the decisions made based on the (test) project's goal and risk assessment. It is the starting point for planning the test process, for selecting the test design techniques and test types to be applied, and for defining the entry and exit criteria.

The selected approach depends on the context and may consider risks, hazards and safety, available resources and skills, the technology, the nature of the system (e.g., custom built vs. COTS), test objectives, and regulations.

Typical approaches include:

- Analytical approaches, such as risk-based testing where testing is directed to areas of greatest risk
- Model-based approaches, such as stochastic testing using statistical information about failure rates (such as reliability growth models) or usage (such as operational profiles)
- Methodical approaches, such as failure-based (including error guessing and fault attacks), experience-based, checklist-based, and quality characteristic-based
- Process- or standard-compliant approaches, such as those specified by industry-specific standards or the various agile methodologies
- Dynamic and heuristic approaches, such as exploratory testing where testing is more reactive to events than pre-planned, and where execution and evaluation are concurrent tasks
- Consultative approaches, such as those in which test coverage is driven primarily by the advice and guidance of technology and/or business domain experts outside the test team
- Regression-averse approaches, such as those that include reuse of existing test material, extensive automation of functional regression tests, and standard test suites

Different approaches may be combined, for example, a risk-based dynamic approach.

5.3 Test Progress Monitoring and Control (K2)

20 minutes

Terms

Defect density, failure rate, test control, test monitoring, test summary report

5.3.1 Test Progress Monitoring (K1)

The purpose of test monitoring is to provide feedback and visibility about test activities. Information to be monitored may be collected manually or automatically and may be used to measure exit criteria, such as coverage. Metrics may also be used to assess progress against the planned schedule and budget. Common test metrics include:

- Percentage of work done in test case preparation (or percentage of planned test cases prepared)
- Percentage of work done in test environment preparation
- Test case execution (e.g., number of test cases run/not run, and test cases passed/failed)
- Defect information (e.g., defect density, defects found and fixed, failure rate, and re-test results)
- Test coverage of requirements, risks or code
- Subjective confidence of testers in the product
- Dates of test milestones
- Testing costs, including the cost compared to the benefit of finding the next defect or to run the next test

5.3.2 Test Reporting (K2)

Test reporting is concerned with summarizing information about the testing endeavor, including:

- What happened during a period of testing, such as dates when exit criteria were met
- Analyzed information and metrics to support recommendations and decisions about future actions, such as an assessment of defects remaining, the economic benefit of continued testing, outstanding risks, and the level of confidence in the tested software

The outline of a test summary report is given in 'Standard for Software Test Documentation' (IEEE Std 829-1998).

Metrics should be collected during and at the end of a test level in order to assess:

- The adequacy of the test objectives for that test level
- The adequacy of the test approaches taken
- The effectiveness of the testing with respect to the objectives

5.3.3 Test Control (K2)

Test control describes any guiding or corrective actions taken as a result of information and metrics gathered and reported. Actions may cover any test activity and may affect any other software life cycle activity or task.

Examples of test control actions include:

- Making decisions based on information from test monitoring
- Re-prioritizing tests when an identified risk occurs (e.g., software delivered late)
- Changing the test schedule due to availability or unavailability of a test environment
- Setting an entry criterion requiring fixes to have been re-tested (confirmation tested) by a developer before accepting them into a build

5.4 Configuration Management (K2)

10 minutes

Terms

Configuration management, version control

Background

The purpose of configuration management is to establish and maintain the integrity of the products (components, data and documentation) of the software or system through the project and product life cycle.

For testing, configuration management may involve ensuring the following:

- All items of testware are identified, version controlled, tracked for changes, related to each other and related to development items (test objects) so that traceability can be maintained throughout the test process
- All identified documents and software items are referenced unambiguously in test documentation

For the tester, configuration management helps to uniquely identify (and to reproduce) the tested item, test documents, the tests and the test harness(es).

During test planning, the configuration management procedures and infrastructure (tools) should be chosen, documented and implemented.

5.5 Risk and Testing (K2)

30 minutes

Terms

Product risk, project risk, risk, risk-based testing

Background

Risk can be defined as the chance of an event, hazard, threat or situation occurring and resulting in undesirable consequences or a potential problem. The level of risk will be determined by the likelihood of an adverse event happening and the impact (the harm resulting from that event).

5.5.1 Project Risks (K2)

Project risks are the risks that surround the project's capability to deliver its objectives, such as:

- Organizational factors:
 - Skill, training and staff shortages
 - Personnel issues
 - Political issues, such as:
 - Problems with testers communicating their needs and test results
 - Failure by the team to follow up on information found in testing and reviews (e.g., not improving development and testing practices)
 - Improper attitude toward or expectations of testing (e.g., not appreciating the value of finding defects during testing)
- Technical issues:
 - Problems in defining the right requirements
 - The extent to which requirements cannot be met given existing constraints
 - Test environment not ready on time
 - Late data conversion, migration planning and development and testing data conversion/migration tools
 - Low quality of the design, code, configuration data, test data and tests
- Supplier issues:
 - Failure of a third party
 - Contractual issues

When analyzing, managing and mitigating these risks, the test manager is following well-established project management principles. The 'Standard for Software Test Documentation' (IEEE Std 829-1998) outline for test plans requires risks and contingencies to be stated.

5.5.2 Product Risks (K2)

Potential failure areas (adverse future events or hazards) in the software or system are known as product risks, as they are a risk to the quality of the product. These include:

- Failure-prone software delivered
- The potential that the software/hardware could cause harm to an individual or company
- Poor software characteristics (e.g., functionality, reliability, usability and performance)
- Poor data integrity and quality (e.g., data migration issues, data conversion problems, data transport problems, violation of data standards)
- Software that does not perform its intended functions

Risks are used to decide where to start testing and where to test more; testing is used to reduce the risk of an adverse effect occurring, or to reduce the impact of an adverse effect.

Product risks are a special type of risk to the success of a project. Testing as a risk-control activity provides feedback about the residual risk by measuring the effectiveness of critical defect removal and of contingency plans.

A risk-based approach to testing provides proactive opportunities to reduce the levels of product risk, starting in the initial stages of a project. It involves the identification of product risks and their use in guiding test planning and control, specification, preparation and execution of tests. In a risk-based approach the risks identified may be used to:

- Determine the test techniques to be employed
- Determine the extent of testing to be carried out
- Prioritize testing in an attempt to find the critical defects as early as possible
- Determine whether any non-testing activities could be employed to reduce risk (e.g., providing training to inexperienced designers)

Risk-based testing draws on the collective knowledge and insight of the project stakeholders to determine the risks and the levels of testing required to address those risks.

To ensure that the chance of a product failure is minimized, risk management activities provide a disciplined approach to:

- Assess (and reassess on a regular basis) what can go wrong (risks)
- Determine what risks are important to deal with
- Implement actions to deal with those risks

In addition, testing may support the identification of new risks, may help to determine what risks should be reduced, and may lower uncertainty about risks.

5.6 Incident Management (K3)

40 minutes

Terms

Incident logging, incident management, incident report

Background

Since one of the objectives of testing is to find defects, the discrepancies between actual and expected outcomes need to be logged as incidents. An incident must be investigated and may turn out to be a defect. Appropriate actions to dispose incidents and defects should be defined. Incidents and defects should be tracked from discovery and classification to correction and confirmation of the solution. In order to manage all incidents to completion, an organization should establish an incident management process and rules for classification.

Incidents may be raised during development, review, testing or use of a software product. They may be raised for issues in code or the working system, or in any type of documentation including requirements, development documents, test documents, and user information such as “Help” or installation guides.

Incident reports have the following objectives:

- Provide developers and other parties with feedback about the problem to enable identification, isolation and correction as necessary
- Provide test leaders a means of tracking the quality of the system under test and the progress of the testing
- Provide ideas for test process improvement

Details of the incident report may include:

- Date of issue, issuing organization, and author
- Expected and actual results
- Identification of the test item (configuration item) and environment
- Software or system life cycle process in which the incident was observed
- Description of the incident to enable reproduction and resolution, including logs, database dumps or screenshots
- Scope or degree of impact on stakeholder(s) interests
- Severity of the impact on the system
- Urgency/priority to fix
- Status of the incident (e.g., open, deferred, duplicate, waiting to be fixed, fixed awaiting re-test, closed)
- Conclusions, recommendations and approvals
- Global issues, such as other areas that may be affected by a change resulting from the incident
- Change history, such as the sequence of actions taken by project team members with respect to the incident to isolate, repair, and confirm it as fixed
- References, including the identity of the test case specification that revealed the problem

The structure of an incident report is also covered in the ‘Standard for Software Test Documentation’ (IEEE Std 829-1998).

6. Tool Support for Testing (K2)	80 minutes
-----------------------------------------	-------------------

Learning Objectives for Tool Support for Testing

The objectives identify what you will be able to do following the completion of each module.

6.1 Types of Test Tools (K2)

- LO-6.1.1 Classify different types of test tools according to their purpose and to the activities of the fundamental test process and the software life cycle (K2)
- LO-6.1.3 Explain the term test tool and the purpose of tool support for testing (K2) ²

6.2 Effective Use of Tools: Potential Benefits and Risks (K2)

- LO-6.2.1 Summarize the potential benefits and risks of test automation and tool support for testing (K2)
- LO-6.2.2 Remember special considerations for test execution tools, static analysis, and test management tools (K1)

6.3 Introducing a Tool into an Organization (K1)

- LO-6.3.1 State the main principles of introducing a tool into an organization (K1)
- LO-6.3.2 State the goals of a proof-of-concept for tool evaluation and a piloting phase for tool implementation (K1)
- LO-6.3.3 Recognize that factors other than simply acquiring a tool are required for good tool support (K1)

6.1 Types of Test Tools (K2)

45 minutes

Terms

Configuration management tool, coverage tool, debugging tool, dynamic analysis tool, incident management tool, load testing tool, modeling tool, monitoring tool, performance testing tool, probe effect, requirements management tool, review tool, security tool, static analysis tool, stress testing tool, test comparator, test data preparation tool, test design tool, test harness, test execution tool, test management tool, unit test framework tool

6.1.1 Tool Support for Testing (K2)

Test tools can be used for one or more activities that support testing. These include:

1. Tools that are directly used in testing such as test execution tools, test data generation tools and result comparison tools
2. Tools that help in managing the testing process such as those used to manage tests, test results, data, requirements, incidents, defects, etc., and for reporting and monitoring test execution
3. Tools that are used in reconnaissance, or, in simple terms: exploration (e.g., tools that monitor file activity for an application)
4. Any tool that aids in testing (a spreadsheet is also a test tool in this meaning)

Tool support for testing can have one or more of the following purposes depending on the context:

- Improve the efficiency of test activities by automating repetitive tasks or supporting manual test activities like test planning, test design, test reporting and monitoring
- Automate activities that require significant resources when done manually (e.g., static testing)
- Automate activities that cannot be executed manually (e.g., large scale performance testing of client-server applications)
- Increase reliability of testing (e.g., by automating large data comparisons or simulating behavior)

The term “test frameworks” is also frequently used in the industry, in at least three meanings:

- Reusable and extensible testing libraries that can be used to build testing tools (called test harnesses as well)
- A type of design of test automation (e.g., data-driven, keyword-driven)
- Overall process of execution of testing

For the purpose of this syllabus, the term “test frameworks” is used in its first two meanings as described in Section 6.1.6.

6.1.2 Test Tool Classification (K2)

There are a number of tools that support different aspects of testing. Tools can be classified based on several criteria such as purpose, commercial / free / open-source / shareware, technology used and so forth. Tools are classified in this syllabus according to the testing activities that they support.

Some tools clearly support one activity; others may support more than one activity, but are classified under the activity with which they are most closely associated. Tools from a single provider, especially those that have been designed to work together, may be bundled into one package.

Some types of test tools can be intrusive, which means that they can affect the actual outcome of the test. For example, the actual timing may be different due to the extra instructions that are executed by the tool, or you may get a different measure of code coverage. The consequence of intrusive tools is called the probe effect.

Some tools offer support more appropriate for developers (e.g., tools that are used during component and component integration testing). Such tools are marked with “(D)” in the list below.

6.1.3 Tool Support for Management of Testing and Tests (K1)

Management tools apply to all test activities over the entire software life cycle.

Test Management Tools

These tools provide interfaces for executing tests, tracking defects and managing requirements, along with support for quantitative analysis and reporting of the test objects. They also support tracing the test objects to requirement specifications and might have an independent version control capability or an interface to an external one.

Requirements Management Tools

These tools store requirement statements, store the attributes for the requirements (including priority), provide unique identifiers and support tracing the requirements to individual tests. These tools may also help with identifying inconsistent or missing requirements.

Incident Management Tools (Defect Tracking Tools)

These tools store and manage incident reports, i.e., defects, failures, change requests or perceived problems and anomalies, and help in managing the life cycle of incidents, optionally with support for statistical analysis.

Configuration Management Tools

Although not strictly test tools, these are necessary for storage and version management of testware and related software especially when configuring more than one hardware/software environment in terms of operating system versions, compilers, browsers, etc.

6.1.4 Tool Support for Static Testing (K1)

Static testing tools provide a cost effective way of finding more defects at an earlier stage in the development process.

Review Tools

These tools assist with review processes, checklists, review guidelines and are used to store and communicate review comments and report on defects and effort. They can be of further help by providing aid for online reviews for large or geographically dispersed teams.

Static Analysis Tools (D)

These tools help developers and testers find defects prior to dynamic testing by providing support for enforcing coding standards (including secure coding), analysis of structures and dependencies. They can also help in planning or risk analysis by providing metrics for the code (e.g., complexity).

Modeling Tools (D)

These tools are used to validate software models (e.g., physical data model (PDM) for a relational database), by enumerating inconsistencies and finding defects. These tools can often aid in generating some test cases based on the model.

6.1.5 Tool Support for Test Specification (K1)

Test Design Tools

These tools are used to generate test inputs or executable tests and/or test oracles from requirements, graphical user interfaces, design models (state, data or object) or code.

Test Data Preparation Tools

Test data preparation tools manipulate databases, files or data transmissions to set up test data to be used during the execution of tests to ensure security through data anonymity.

6.1.6 Tool Support for Test Execution and Logging (K1)

Test Execution Tools

These tools enable tests to be executed automatically, or semi-automatically, using stored inputs and expected outcomes, through the use of a scripting language and usually provide a test log for each test run. They can also be used to record tests, and usually support scripting languages or GUI-based configuration for parameterization of data and other customization in the tests.

Test Harness/Unit Test Framework Tools (D)

A unit test harness or framework facilitates the testing of components or parts of a system by simulating the environment in which that test object will run, through the provision of mock objects as stubs or drivers.

Test Comparators

Test comparators determine differences between files, databases or test results. Test execution tools typically include dynamic comparators, but post-execution comparison may be done by a separate comparison tool. A test comparator may use a test oracle, especially if it is automated.

Coverage Measurement Tools (D)

These tools, through intrusive or non-intrusive means, measure the percentage of specific types of code structures that have been exercised (e.g., statements, branches or decisions, and module or function calls) by a set of tests.

Security Testing Tools

These tools are used to evaluate the security characteristics of software. This includes evaluating the ability of the software to protect data confidentiality, integrity, authentication, authorization, availability, and non-repudiation. Security tools are mostly focused on a particular technology, platform, and purpose.

6.1.7 Tool Support for Performance and Monitoring (K1)

Dynamic Analysis Tools (D)

Dynamic analysis tools find defects that are evident only when software is executing, such as time dependencies or memory leaks. They are typically used in component and component integration testing, and when testing middleware.

Performance Testing/Load Testing/Stress Testing Tools

Performance testing tools monitor and report on how a system behaves under a variety of simulated usage conditions in terms of number of concurrent users, their ramp-up pattern, frequency and relative percentage of transactions. The simulation of load is achieved by means of creating virtual users carrying out a selected set of transactions, spread across various test machines commonly known as load generators.

Monitoring Tools

Monitoring tools continuously analyze, verify and report on usage of specific system resources, and give warnings of possible service problems.

6.1.8 Tool Support for Specific Testing Needs (K1)

Data Quality Assessment

Data is at the center of some projects such as data conversion/migration projects and applications like data warehouses and its attributes can vary in terms of criticality and volume. In such contexts, tools need to be employed for data quality assessment to review and verify the data conversion and

migration rules to ensure that the processed data is correct, complete and complies with a pre-defined context-specific standard.

Other testing tools exist for usability testing.

6.2 Effective Use of Tools: Potential Benefits and Risks (K2)

20 minutes

Terms

Data-driven testing, keyword-driven testing, scripting language

6.2.1 Potential Benefits and Risks of Tool Support for Testing (for all tools) (K2)

Simply purchasing or leasing a tool does not guarantee success with that tool. Each type of tool may require additional effort to achieve real and lasting benefits. There are potential benefits and opportunities with the use of tools in testing, but there are also risks.

Potential benefits of using tools include:

- Repetitive work is reduced (e.g., running regression tests, re-entering the same test data, and checking against coding standards)
- Greater consistency and repeatability (e.g., tests executed by a tool in the same order with the same frequency, and tests derived from requirements)
- Objective assessment (e.g., static measures, coverage)
- Ease of access to information about tests or testing (e.g., statistics and graphs about test progress, incident rates and performance)

Risks of using tools include:

- Unrealistic expectations for the tool (including functionality and ease of use)
- Underestimating the time, cost and effort for the initial introduction of a tool (including training and external expertise)
- Underestimating the time and effort needed to achieve significant and continuing benefits from the tool (including the need for changes in the testing process and continuous improvement of the way the tool is used)
- Underestimating the effort required to maintain the test assets generated by the tool
- Over-reliance on the tool (replacement for test design or use of automated testing where manual testing would be better)
- Neglecting version control of test assets within the tool
- Neglecting relationships and interoperability issues between critical tools, such as requirements management tools, version control tools, incident management tools, defect tracking tools and tools from multiple vendors
- Risk of tool vendor going out of business, retiring the tool, or selling the tool to a different vendor
- Poor response from vendor for support, upgrades, and defect fixes
- Risk of suspension of open-source / free tool project
- Unforeseen, such as the inability to support a new platform

6.2.2 Special Considerations for Some Types of Tools (K1)

Test Execution Tools

Test execution tools execute test objects using automated test scripts. This type of tool often requires significant effort in order to achieve significant benefits.

Capturing tests by recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of automated test scripts. A captured script is a linear representation with specific data and actions as part of each script. This type of script may be unstable when unexpected events occur.

A data-driven testing approach separates out the test inputs (the data), usually into a spreadsheet, and uses a more generic test script that can read the input data and execute the same test script with different data. Testers who are not familiar with the scripting language can then create the test data for these predefined scripts.

There are other techniques employed in data-driven techniques, where instead of hard-coded data combinations placed in a spreadsheet, data is generated using algorithms based on configurable parameters at run time and supplied to the application. For example, a tool may use an algorithm, which generates a random user ID, and for repeatability in pattern, a seed is employed for controlling randomness.

In a keyword-driven testing approach, the spreadsheet contains keywords describing the actions to be taken (also called action words), and test data. Testers (even if they are not familiar with the scripting language) can then define tests using the keywords, which can be tailored to the application being tested.

Technical expertise in the scripting language is needed for all approaches (either by testers or by specialists in test automation).

Regardless of the scripting technique used, the expected results for each test need to be stored for later comparison.

Static Analysis Tools

Static analysis tools applied to source code can enforce coding standards, but if applied to existing code may generate a large quantity of messages. Warning messages do not stop the code from being translated into an executable program, but ideally should be addressed so that maintenance of the code is easier in the future. A gradual implementation of the analysis tool with initial filters to exclude some messages is an effective approach.

Test Management Tools

Test management tools need to interface with other tools or spreadsheets in order to produce useful information in a format that fits the needs of the organization.

6.3 Introducing a Tool into an Organization (K1)

15 minutes

Terms

No specific terms.

Background

The main considerations in selecting a tool for an organization include:

- Assessment of organizational maturity, strengths and weaknesses and identification of opportunities for an improved test process supported by tools
- Evaluation against clear requirements and objective criteria
- A proof-of-concept, by using a test tool during the evaluation phase to establish whether it performs effectively with the software under test and within the current infrastructure or to identify changes needed to that infrastructure to effectively use the tool
- Evaluation of the vendor (including training, support and commercial aspects) or service support suppliers in case of non-commercial tools
- Identification of internal requirements for coaching and mentoring in the use of the tool
- Evaluation of training needs considering the current test team's test automation skills
- Estimation of a cost-benefit ratio based on a concrete business case

Introducing the selected tool into an organization starts with a pilot project, which has the following objectives:

- Learn more detail about the tool
- Evaluate how the tool fits with existing processes and practices, and determine what would need to change
- Decide on standard ways of using, managing, storing and maintaining the tool and the test assets (e.g., deciding on naming conventions for files and tests, creating libraries and defining the modularity of test suites)
- Assess whether the benefits will be achieved at reasonable cost

Success factors for the deployment of the tool within an organization include:

- Rolling out the tool to the rest of the organization incrementally
- Adapting and improving processes to fit with the use of the tool
- Providing training and coaching/mentoring for new users
- Defining usage guidelines
- Implementing a way to gather usage information from the actual use
- Monitoring tool use and benefits
- Providing support for the test team for a given tool
- Gathering lessons learned from all teams

References

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

9. Appendix B – Learning Objectives/Cognitive Level of Knowledge

The following learning objectives are defined as applying to this syllabus. Each topic in the syllabus will be examined according to the learning objective for it.

Level 1: Remember (K1)

The candidate will recognize, remember and recall a term or concept.

Keywords: Remember, retrieve, recall, recognize, know

Example

Can recognize the definition of “failure” as:

- “Non-delivery of service to an end user or any other stakeholder” or
- “Actual deviation of the component or system from its expected delivery, service or result”

Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify, categorize and give examples for the testing concept.

Keywords: Summarize, generalize, abstract, classify, compare, map, contrast, exemplify, interpret, translate, represent, infer, conclude, categorize, construct models

Examples

Can explain the reason why tests should be designed as early as possible:

- To find defects when they are cheaper to remove
- To find the most important defects first

Can explain the similarities and differences between integration and system testing:

- Similarities: testing more than one component, and can test non-functional aspects
- Differences: integration testing concentrates on interfaces and interactions, and system testing concentrates on whole-system aspects, such as end-to-end processing

Level 3: Apply (K3)

The candidate can select the correct application of a concept or technique and apply it to a given context.

Keywords: Implement, execute, use, follow a procedure, apply a procedure

Example

- Can identify boundary values for valid and invalid partitions
- Can select test cases from a given state transition diagram in order to cover all transitions

Level 4: Analyze (K4)

The candidate can separate information related to a procedure or technique into its constituent parts for better understanding, and can distinguish between facts and inferences. Typical application is to analyze a document, software or project situation and propose appropriate actions to solve a problem or task.

Keywords: Analyze, organize, find coherence, integrate, outline, parse, structure, attribute, deconstruct, differentiate, discriminate, distinguish, focus, select

Example

- Analyze product risks and propose preventive and corrective mitigation activities
- Describe which portions of an incident report are factual and which are inferred from results

Reference

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon

10. Appendix C – Rules Applied to the ISTQB

Foundation Syllabus

The rules listed here were used in the development and review of this syllabus. (A “TAG” is shown after each rule as a shorthand abbreviation of the rule.)

10.1.1 General Rules

- SG1. The syllabus should be understandable and absorbable by people with zero to six months (or more) experience in testing. (6-MONTH)
- SG2. The syllabus should be practical rather than theoretical. (PRACTICAL)
- SG3. The syllabus should be clear and unambiguous to its intended readers. (CLEAR)
- SG4. The syllabus should be understandable to people from different countries, and easily translatable into different languages. (TRANSLATABLE)
- SG5. The syllabus should use American English. (AMERICAN-ENGLISH)

10.1.2 Current Content

- SC1. The syllabus should include recent testing concepts and should reflect current best practices in software testing where this is generally agreed. The syllabus is subject to review every three to five years. (RECENT)
- SC2. The syllabus should minimize time-related issues, such as current market conditions, to enable it to have a shelf life of three to five years. (SHELF-LIFE).

10.1.3 Learning Objectives

- LO1. Learning objectives should distinguish between items to be recognized/remembered (cognitive level K1), items the candidate should understand conceptually (K2), items the candidate should be able to practice/use (K3), and items the candidate should be able to use to analyze a document, software or project situation in context (K4). (KNOWLEDGE-LEVEL)
- LO2. The description of the content should be consistent with the learning objectives. (LO-CONSISTENT)
- LO3. To illustrate the learning objectives, sample exam questions for each major section should be issued along with the syllabus. (LO-EXAM)

10.1.4 Overall Structure

- ST1. The structure of the syllabus should be clear and allow cross-referencing to and from other parts, from exam questions and from other relevant documents. (CROSS-REF)
- ST2. Overlap between sections of the syllabus should be minimized. (OVERLAP)
- ST3. Each section of the syllabus should have the same structure. (STRUCTURE-CONSISTENT)
- ST4. The syllabus should contain version, date of issue and page number on every page. (VERSION)
- ST5. The syllabus should include a guideline for the amount of time to be spent in each section (to reflect the relative importance of each topic). (TIME-SPENT)

References

- SR1. Sources and references will be given for concepts in the syllabus to help training providers find out more information about the topic. (REFS)
- SR2. Where there are not readily identified and clear sources, more detail should be provided in the syllabus. For example, definitions are in the Glossary, so only the terms are listed in the syllabus. (NON-REF DETAIL)

11. Appendix D – Notice to Training Providers

Each major subject heading in the syllabus is assigned an allocated time in minutes. The purpose of this is both to give guidance on the relative proportion of time to be allocated to each section of an accredited course, and to give an approximate minimum time for the teaching of each section. Training providers may spend more time than is indicated and candidates may spend more time again in reading and research. A course curriculum does not have to follow the same order as the syllabus.

The syllabus contains references to established standards, which must be used in the preparation of training material. Each standard used must be the version quoted in the current version of this syllabus. Other publications, templates or standards not referenced in this syllabus may also be used and referenced, but will not be examined.

All K3 and K4 Learning Objectives require a practical exercise to be included in the training materials.

12. Appendix E – Release Notes

Release 2010

1. Changes to Learning Objectives (LO) include some clarification
 - a. Wording changed for the following LOs (content and level of LO remains unchanged): LO-1.2.2, LO-1.3.1, LO-1.4.1, LO-1.5.1, LO-2.1.1, LO-2.1.3, LO-2.4.2, LO-4.1.3, LO-4.2.1, LO-4.2.2, LO-4.3.1, LO-4.3.2, LO-4.3.3, LO-4.4.1, LO-4.4.2, LO-4.4.3, LO-4.6.1, LO-5.1.2, LO-5.2.2, LO-5.3.2, LO-5.3.3, LO-5.5.2, LO-5.6.1, LO-6.1.1, LO-6.2.2, LO-6.3.2.
 - b. LO-1.1.5 has been reworded and upgraded to K2. Because a comparison of terms of defect related terms can be expected.
 - c. LO-1.2.3 (K2) has been added. The content was already covered in the 2007 syllabus.
 - d. LO-3.1.3 (K2) now combines the content of LO-3.1.3 and LO-3.1.4.
 - e. LO-3.1.4 has been removed from the 2010 syllabus, as it is partially redundant with LO-3.1.3.
 - f. LO-3.2.1 has been reworded for consistency with the 2010 syllabus content.
 - g. LO-3.3.2 has been modified, and its level has been changed from K1 to K2, for consistency with LO-3.1.2.
 - h. LO 4.4.4 has been modified for clarity, and has been changed from a K3 to a K4. Reason: LO-4.4.4 had already been written in a K4 manner.
 - i. LO-6.1.2 (K1) was dropped from the 2010 syllabus and was replaced with LO-6.1.3 (K2). There is no LO-6.1.2 in the 2010 syllabus.
2. Consistent use for test approach according to the definition in the glossary. The term test strategy will not be required as term to recall.
3. Chapter 1.4 now contains the concept of traceability between test basis and test cases.
4. Chapter 2.x now contains test objects and test basis.
5. Re-testing is now the main term in the glossary instead of confirmation testing.
6. The aspect data quality and testing has been added at several locations in the syllabus: data quality and risk in Chapter 2.2, 5.5, 6.1.8.
7. Chapter 5.2.3 Entry Criteria are added as a new subchapter. Reason: Consistency to Exit Criteria (-> entry criteria added to LO-5.2.9).
8. Consistent use of the terms test strategy and test approach with their definition in the glossary.
9. Chapter 6.1 shortened because the tool descriptions were too large for a 45 minute lesson.
10. IEEE Std 829:2008 has been released. This version of the syllabus does not yet consider this new edition. Section 5.2 refers to the document Master Test Plan. The content of the Master Test Plan is covered by the concept that the document "Test Plan" covers different levels of planning: Test plans for the test levels can be created as well as a test plan on the project level covering multiple test levels. Latter is named Master Test Plan in this syllabus and in the ISTQB Glossary.
11. Code of Ethics has been moved from the CTAL to CTFL.

Release 2011

Changes made with the "maintenance release" 2011

1. General: Working Party replaced by Working Group
2. Replaced post-conditions by postconditions in order to be consistent with the ISTQB Glossary 2.1.
3. First occurrence: ISTQB replaced by ISTQB®
4. Introduction to this Syllabus: Descriptions of Cognitive Levels of Knowledge removed, because this was redundant to Appendix B.

5. Section 1.6: Because the intent was not to define a Learning Objective for the “Code of Ethics”, the cognitive level for the section has been removed.
6. Section 2.2.1, 2.2.2, 2.2.3 and 2.2.4, 3.2.3: Fixed formatting issues in lists.
7. Section 2.2.2 The word failure was not correct for “...isolate failures to a specific component ...”. Therefore replaced with “defect” in that sentence.
8. Section 2.3: Corrected formatting of bullet list of test objectives related to test terms in section Test Types (K2).
9. Section 2.3.4: Updated description of debugging to be consistent with Version 2.1 of the ISTQB Glossary.
10. Section 2.4 removed word “extensive” from “includes extensive regression testing”, because the “extensive” depends on the change (size, risks, value, etc.) as written in the next sentence.
11. Section 3.2: The word “including” has been removed to clarify the sentence.
12. Section 3.2.1: Because the activities of a formal review had been incorrectly formatted, the review process had 12 main activities instead of six, as intended. It has been changed back to six, which makes this section compliant with the Syllabus 2007 and the ISTQB Advanced Level Syllabus 2007.
13. Section 4: Word “developed” replaced by “defined” because test cases get defined and not developed.
14. Section 4.2: Text change to clarify how black-box and white-box testing could be used in conjunction with experience-based techniques.
15. Section 4.3.5 text change “..between actors, including users and the system..” to “ ... between actors (users or systems), ... “.
16. Section 4.3.5 alternative path replaced by alternative scenario.
17. Section 4.4.2: In order to clarify the term branch testing in the text of Section 4.4, a sentence to clarify the focus of branch testing has been changed.
18. Section 4.5, Section 5.2.6: The term “experienced-based” testing has been replaced by the correct term “experience-based”.
19. Section 6.1: Heading “6.1.1 Understanding the Meaning and Purpose of Tool Support for Testing (K2)” replaced by “6.1.1 Tool Support for Testing (K2)”.
20. Section 7 / Books: The 3rd edition of [Black,2001] listed, replacing 2nd edition.
21. Appendix D: Chapters requiring exercises have been replaced by the generic requirement that all Learning Objectives K3 and higher require exercises. This is a requirement specified in the ISTQB Accreditation Process (Version 1.26).
22. Appendix E: The changed learning objectives between Version 2007 and 2010 are now correctly listed.

13. Index

- action word 63
- alpha testing 24, 27
- architecture 15, 21, 22, 25, 28, 29
- archiving 17, 30
- automation 29
- benefits of independence 47
- benefits of using tool 62
- beta testing 24, 27
- black-box technique 37, 39, 40
- black-box test design technique 39
- black-box testing 28
- bottom-up 25
- boundary value analysis 40
- bug 11
- captured script 62
- checklists 34, 35
- choosing test technique 44
- code coverage 28, 29, 37, 42, 58
- commercial off the shelf (COTS) 22
- compiler 36
- complexity 11, 36, 50, 59
- component integration testing 22, 25, 29, 59, 60
- component testing 22, 24, 25, 27, 29, 37, 41, 42
- configuration management 45, 48, 52
- Configuration management tool 58
- confirmation testing 13, 15, 16, 21, 28, 29
- contract acceptance testing 27
- control flow 28, 36, 37, 42
- coverage 15, 24, 28, 29, 37, 38, 39, 40, 42, 50, 51, 58, 60, 62
- coverage tool 58
- custom-developed software 27
- data flow 36
- data-driven approach 63
- data-driven testing 62
- debugging 13, 24, 29, 58
- debugging tool 24, 58
- decision coverage 37, 42
- decision table testing 40, 41
- decision testing 42
- defect 10, 11, 13, 14, 16, 18, 21, 24, 26, 28, 29, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 43, 44, 45, 47, 49, 50, 51, 53, 54, 55, 59, 60, 69
- defect density 50, 51
- defect tracking tool 59
- development 8, 11, 12, 13, 14, 18, 21, 22, 24, 29, 32, 33, 36, 38, 44, 47, 49, 50, 52, 53, 55, 59, 67
- development model 21, 22
- drawbacks of independence 47
- driver 24
- dynamic analysis tool 58, 60
- dynamic testing 13, 31, 32, 36
- emergency change 30
- enhancement 27, 30
- entry criteria 33
- equivalence partitioning 40
- error 10, 11, 18, 43, 50
- error guessing 18, 43, 50
- exhaustive testing 14
- exit criteria 13, 15, 16, 33, 35, 45, 48, 49, 50, 51
- expected result 16, 38, 48, 63
- experience-based technique 37, 39, 43
- experience-based test design technique 39
- exploratory testing 43, 50
- factory acceptance testing 27
- failure 10, 11, 13, 14, 18, 21, 24, 26, 32, 36, 43, 46, 50, 51, 53, 54, 69
- failure rate 50, 51
- fault 10, 11, 43
- fault attack 43
- field testing 24, 27
- follow-up 33, 34, 35
- formal review 31, 33
- functional requirement 24, 26
- functional specification 28
- functional task 25
- functional test 28
- functional testing 28
- functionality 24, 25, 28, 50, 53, 62
- impact analysis 21, 30, 38
- incident 15, 16, 17, 19, 24, 46, 48, 55, 58, 59, 62
- incident logging 55
- incident management 48, 55, 58
- incident management tool 58, 59
- incident report 46, 55
- independence 18, 47, 48
- informal review 31, 33, 34
- inspection 31, 33, 34, 35
- inspection leader 33
- integration 13, 22, 24, 25, 27, 29, 36, 40, 41, 42, 45, 48, 59, 60, 69
- integration testing 22, 24, 25, 29, 36, 40, 45, 59, 60, 69
- interoperability testing 28
- introducing a tool into an organization 57, 64
- ISO 9126 11, 29, 30, 65
- development model 22
- iterative-incremental development model 22
- keyword-driven approach 63
- keyword-driven testing 62
- kick-off 33
- learning objective 8, 9, 10, 21, 31, 37, 45, 57, 69, 70, 71
- load testing 28, 58, 60

load testing tool.....	58
maintainability testing	28
maintenance testing	21, 30
management tool.....	48, 58, 59, 63
maturity	17, 33, 38, 64
metric	33, 35, 45
mistake	10, 11, 16
modelling tool.....	59
moderator	33, 34, 35
monitoring tool	48, 58
non-functional requirement.....	21, 24, 26
non-functional testing	11, 28
objectives for testing.....	13
off-the-shelf	22
operational acceptance testing.....	27
operational test	13, 23, 30
patch	30
peer review	33, 34, 35
performance testing	28, 58
performance testing tool	58, 60
pesticide paradox.....	14
portability testing.....	28
probe effect.....	58
procedure.....	16
product risk	18, 45, 53, 54
project risk	12, 45, 53
prototyping	22
quality 8, 10, 11, 13, 19, 28, 37, 38, 47, 48, 50, 53, 55, 59	
rapid application development (RAD).....	22
Rational Unified Process (RUP)	22
recorder	34
regression testing	15, 16, 21, 28, 29, 30
Regulation acceptance testing	27
reliability.....	11, 13, 28, 50, 53, 58
reliability testing	28
requirement.....	13, 22, 24, 32, 34
requirements management tool.....	58
requirements specification.....	26, 28
responsibilities	24, 31, 33
re-testing. 29. See confirmation testing. See confirmation testing	
review13, 19, 31, 32, 33, 34, 35, 36, 47, 48, 53, 55, 58, 67, 71	
review tool.....	58
reviewer	33, 34
risk11, 12, 13, 14, 25, 26, 29, 30, 38, 44, 45, 49, 50, 51, 53, 54	
risk-based approach	54
risk-based testing.....	50, 53, 54
risks	11, 25, 49, 53
risks of using tool.....	62
robustness testing.....	24
roles	8, 31, 33, 34, 35, 47, 48, 49
root cause	10, 11
scribe	33, 34
scripting language.....	60, 62, 63
security	27, 28, 36, 47, 50, 58
security testing	28
security tool	58, 60
simulators	24
site acceptance testing	27
software development.....	8, 11, 21, 22
software development model	22
special considerations for some types of tool62	
test case.....	38
specification-based technique.....	29, 39, 40
specification-based testing.....	37
stakeholders..12, 13, 16, 18, 26, 39, 45, 54	
state transition testing	40, 41
statement coverage	42
statement testing.....	42
static analysis.....	32, 36
static analysis tool.....	31, 36, 58, 59, 63
static technique	31, 32
static testing	13, 32
stress testing	28, 58, 60
stress testing tool	58, 60
structural testing.....	24, 28, 29, 42
structure-based technique	39, 42
structure-based test design technique....	42
structure-based testing	37, 42
stub	24
success factors	35
system integration testing	22, 25
system testing13, 22, 24, 25, 26, 27, 49, 69	
technical review	31, 33, 34, 35
test analysis	15, 38, 48, 49
test approach	38, 48, 50, 51
test basis	15
test case.13, 14, 15, 16, 24, 28, 32, 37, 38, 39, 40, 41, 42, 45, 51, 55, 59, 69	
test case specification.....	37, 38, 55
test cases	28
test closure.....	10, 15, 16
test condition	38
test conditions	13, 15, 16, 28, 38, 39
test control.....	15, 45, 51
test coverage	15, 50
test data	15, 16, 38, 48, 58, 60, 62, 63
test data preparation tool	58, 60
test design13, 15, 22, 37, 38, 39, 43, 48, 58, 62	
test design specification.....	45
test design technique	37, 38, 39
test design tool.....	58, 59
Test Development Process.....	38
test effort	50
test environment .15, 16, 17, 24, 26, 48, 51	
test estimation	50
test execution13, 15, 16, 32, 36, 38, 43, 45, 57, 58, 60	
test execution schedule	38
test execution tool	16, 38, 57, 58, 60, 62
test harness.....	16, 24, 52, 58, 60
test implementation	16, 38, 49

test leader	18, 45, 47, 55
test leader tasks.....	47
test level. 21, 22, 24, 28, 29, 30, 37, 40, 42, 44, 45, 48, 49	
test log	15, 16, 43, 60
test management.....	45, 58
test management tool	58, 63
test manager.....	8, 47, 53
test monitoring	48, 51
test objective.....	13, 22, 28, 43, 44, 48, 51
test oracle	60
test organization	47
test plan ..	15, 16, 32, 45, 48, 49, 52, 53, 54
test planning	15, 16, 45, 49, 52, 54
test planning activities	49
test procedure.....	15, 16, 37, 38, 45, 49
test procedure specification	37, 38
test progress monitoring	51
test report.....	45, 51
test reporting.....	45, 51
test script	16, 32, 38
test strategy	47
test suite	29
test summary report.....	15, 16, 45, 48, 51
test tool classification.....	58
test type	21, 28, 30, 48, 75
test-driven development	24
tester 10, 13, 18, 34, 41, 43, 45, 47, 48, 52, 62, 67	
tester tasks	48
test-first approach	24
testing and quality	11
testing principles	10, 14
testware.....	15, 16, 17, 48, 52
tool support	24, 32, 42, 57, 62
tool support for management of testing and tests	59
tool support for performance and monitoring	60
tool support for static testing	59
tool support for test execution and logging	60
tool support for test specification	59
tool support for testing	57, 62
top-down	25
traceability	38, 48, 52
transaction processing sequences	25
types of test tool	57, 58
unit test framework.....	24, 58, 60
unit test framework tool.....	58, 60
upgrades	30
usability	11, 27, 28, 45, 47, 53
usability testing	28, 45
use case test.....	37, 40
use case testing	37, 40, 41
use cases	22, 26, 28, 41
user acceptance testing	27
validation	22
verification	22
version control.....	52
V-model.....	22
walkthrough.....	31, 33, 34
white-box test design technique	39, 42
white-box testing	28, 42

Appendix B

Glossary

Standard Glossary of Terms used in Software Testing

Version 3.0

acceptance criteria

Ref: IEEE 610

The exit criteria that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity.

acceptance testing

Ref: After IEEE 610 **See Also:** user acceptance testing

Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

actor

User or any other person or system that interacts with the test object in a specific way.

actual result

Synonyms: actual outcome

The behavior produced/observed when a component or system is tested.

alpha testing

Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site, but outside the development organization. Alpha testing is often employed for commercial off-the-shelf software as a form of internal acceptance testing.

audit

Ref: IEEE 1028

An independent evaluation of software products or processes to ascertain compliance to standards, guidelines, specifications, and/or procedures based on objective criteria, including documents that specify: the form or content of the products to be produced, the process by which the products shall be produced, and how compliance to standards or guidelines shall be measured.

audit trail

Ref: After TMap

A path by which the original input to a process (e.g., data) can be traced back through the process, taking the process output as a starting point. This facilitates defect analysis and allows a process audit to be carried out.

availability

Ref: IEEE 610

The degree to which a component or system is operational and accessible when required for use. Often expressed as a percentage.

best practice

A superior method or innovative practice that contributes to the improved performance of an organization under given context, usually recognized as "best" by other peer organizations.

beta testing

Synonyms: field testing

Operational testing by potential and/or existing users/customers at an external site not otherwise involved with the developers, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes. Beta testing is often employed as a form of external acceptance testing for commercial off-the-shelf software in order to acquire feedback from the market.

black-box test design technique

Synonyms: black-box technique , specification-based technique , specification-based test design technique

Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

black-box testing

Synonyms: specification-based testing

Testing, either functional or non-functional, without reference to the internal structure of the component or system.

boundary value

An input value or output value which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum or maximum value of a range.

boundary value analysis

See Also: boundary value

A black-box test design technique in which test cases are designed based on boundary values.

branch

A basic block that can be selected for execution based on a program construct in which one of two or more alternative program paths is available, e.g., case, jump, go to, if-then-else.

branch testing

A white-box test design technique in which test cases are designed to execute branches.

Capability Maturity Model Integration (CMMI)

Ref: CMMI

A framework that describes the key elements of an effective product development and maintenance process. The Capability Maturity Model Integration covers best-practices for planning, engineering and managing product development and maintenance.

certification

The process of confirming that a component, system or person complies with its specified requirements, e.g., by passing an exam.

checklist-based testing

An experience-based test design technique whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified.

code

Ref: IEEE 610

Computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler or other translator.

code coverage

An analysis method that determines which parts of the software have been executed (covered) by the test suite and which parts have not been executed, e.g., statement coverage, decision coverage or condition coverage.

commercial off-the-shelf (COTS)

Synonyms: off-the-shelf software

A software product that is developed for the general market, i.e. for a large number of customers, and that is delivered to many customers in identical format.

compiler

Ref: IEEE 610

A software tool that translates programs expressed in a high-order language into their machine language equivalents.

complexity

See Also: cyclomatic complexity

The degree to which a component or system has a design and/or internal structure that is difficult to understand, maintain and verify.

compliance

Ref: ISO 9126

The capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions.

component

Synonyms: module , unit

A minimal software item that can be tested in isolation.

component integration testing

Synonyms: link testing

Testing performed to expose defects in the interfaces and interaction between integrated components.

component specification

A description of a component's function in terms of its output values for specified input values under specified conditions, and required non-functional behavior (e.g., resource-utilization).

component testing

Ref: After IEEE 610

Synonyms: module testing , program testing , unit testing

The testing of individual software components.

condition

See Also: condition testing

Synonyms: branch condition

A logical expression that can be evaluated as True or False, e.g., $A > B$.

condition coverage

Synonyms: branch condition coverage

The percentage of condition outcomes that have been exercised by a test suite. 100% condition coverage requires each single condition in every decision statement to be tested as True and False.

configuration

The composition of a component or system as defined by the number, nature, and interconnections of its constituent parts.

configuration control

Ref: IEEE 610

Synonyms: change control , version control

An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.

configuration item

Ref: IEEE 610

An aggregation of hardware, software or both, that is designated for configuration management and treated as a single entity in the configuration management process.

configuration management

Ref: IEEE 610

A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

configuration management tool

A tool that provides support for the identification and control of configuration items, their status over changes and versions, and the release of baselines consisting of configuration items.

confirmation testing

Synonyms: re-testing

Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

control flow

A sequence of events (paths) in the execution through a component or system.

control flow testing

See Also: decision testing, condition testing, path testing

An approach to structure-based testing in which test cases are designed to execute specific sequences of events. Various techniques exist for control flow testing, e.g., decision testing, condition testing, and path testing, that each have their specific approach and level of control flow coverage.

conversion testing

Synonyms: migration testing

Testing of software used to convert data from existing systems for use in replacement systems.

coverage

Synonyms: test coverage

The degree, expressed as a percentage, to which a specified coverage item has been exercised by a test suite.

coverage tool

Synonyms: coverage measurement tool

A tool that provides objective measures of what structural elements, e.g., statements, branches have been exercised by a test suite.

data flow

Ref: Beiser

An abstract representation of the sequence and possible changes of the state of data objects, where the state of an object is any of creation, usage, or destruction.

data quality

An attribute of data that indicates correctness with respect to some pre-defined criteria, e.g., business expectations, requirements on data integrity, data consistency.

data-driven testing

Ref: Fewster and Graham **See Also:** keyword-driven testing

A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools.

debugging

The process of finding, analyzing and removing the causes of failures in software.

debugging tool

Synonyms: debugger

A tool used by programmers to reproduce failures, investigate the state of programs and find the corresponding defect. Debuggers enable programmers to execute programs step by step, to halt a program at any program statement and to set and examine program variables.

decision

A program point at which the control flow has two or more alternative routes. A node with two or more links to separate branches.

decision coverage

The percentage of decision outcomes that have been exercised by a test suite. 100% decision coverage implies both 100% branch coverage and 100% statement coverage.

decision outcome

The result of a decision (which therefore determines the branches to be taken).

decision table

Synonyms: cause-effect decision table

A table showing combinations of inputs and/or stimuli (causes) with their associated outputs and/or actions (effects), which can be used to design test cases.

decision table testing

Ref: Egler63 **See Also:** decision table

A black-box test design technique in which test cases are designed to execute the combinations of inputs and/or stimuli (causes) shown in a decision table.

decision testing

A white-box test design technique in which test cases are designed to execute decision outcomes.

defect

Synonyms: bug , fault , problem

A flaw in a component or system that can cause the component or system to fail to perform its required function, e.g., an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system.

defect density

Synonyms: fault density

The number of defects identified in a component or system divided by the size of the component or system (expressed in standard measurement terms, e.g., lines-of-code, number of classes or function points).

defect management tool

See Also: incident management tool

Synonyms: bug tracking tool , defect tracking tool

A tool that facilitates the recording and status tracking of defects and changes. They often have workflow-oriented facilities to track and control the allocation, correction and re-testing of defects and provide reporting facilities.

defect type

Synonyms: defect category

An element in a taxonomy of defects. Defect taxonomies can be identified with respect to a variety of considerations, including, but not limited to: Phase or development activity in which the defect is created, e.g., a specification error or a coding error, Characterization of defects, e.g., an "off-by-one" defect, Incorrectness, e.g., an incorrect relational operator, a programming language syntax error, or an invalid assumption, Performance issues, e.g., excessive execution time, insufficient availability.

deliverable

Any (work) product that must be delivered to someone other than the (work) product's author.

development testing

Ref: After IEEE 610

Formal or informal testing conducted during the implementation of a component or system, usually in the development environment by developers.

domain

The set from which valid input and/or output values can be selected.

driver

Ref: After TMap

Synonyms: test driver

A software component or test tool that replaces a component that takes care of the control and/or the calling of a component or system.

dynamic analysis tool

A tool that provides run-time information on the state of the software code. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic and to monitor the allocation, use and de-allocation of memory and to flag memory leaks.

dynamic testing

Testing that involves the execution of the software of a component or system.

effectiveness

See Also: efficiency

The capability of producing an intended result.

efficiency

Ref: ISO 9126

(1) The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. (2) The capability of a process to produce the intended outcome, relative to the amount of resources used.

entry criteria

Ref: Gilb and Graham

The set of generic and specific conditions for permitting a process to go forward with a defined task, e.g., test phase. The purpose of entry criteria is to prevent a task from starting which would entail more (wasted) effort compared to the effort needed to remove the failed entry criteria.

equivalence partition

Synonyms: equivalence class

A portion of an input or output domain for which the behavior of a component or system is assumed to be the same, based on the specification.

equivalence partitioning

Synonyms: partition testing

A black-box test design technique in which test cases are designed to execute representatives from equivalence partitions. In principle, test cases are designed to cover each partition at least once.

error

Ref: After IEEE 610

Synonyms: mistake

A human action that produces an incorrect result.

error guessing

A test design technique where the experience of the tester is used to anticipate what defects might be present in the component or system under test as a result of errors made, and to design tests specifically to expose them.

executable statement

A statement which, when compiled, is translated into object code, and which will be executed procedurally when the program is running and may perform an action on data.

exercised

A program element is said to be exercised by a test case when the input value causes the execution of that element, such as a statement, decision, or other structural element.

exhaustive testing

Synonyms: complete testing

A test approach in which the test suite comprises all combinations of input values and preconditions.

exit criteria

Ref: After Gilb and Graham

Synonyms: completion criteria , test completion criteria

The set of generic and specific conditions, agreed upon with the stakeholders for permitting a process to be officially completed. The purpose of exit criteria is to prevent a task from being considered completed when there are still outstanding parts of the task which have not been finished. Exit criteria are used to report against and to plan when to stop testing.

expected result

Synonyms: expected outcome , predicted outcome

The behavior predicted by the specification, or another source, of the component or system under specified conditions.

experience-based test design technique

Synonyms: experience-based technique

Procedure to derive and/or select test cases based on the tester's experience, knowledge and intuition.

experience-based testing

Testing based on the tester's experience, knowledge and intuition.

exploratory testing

Ref: After Bach

An informal test design technique where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.

factory acceptance testing

See Also: alpha testing

Acceptance testing conducted at the site at which the product is developed and performed by employees of the supplier organization, to determine whether or not a component or system satisfies the requirements, normally including hardware as well as software.

fail

Synonyms: test fail

A test is deemed to fail if its actual result does not match its expected result.

failure

Ref: After Fenton

Deviation of the component or system from its expected delivery, service or result.

failure rate

Ref: IEEE 610

The ratio of the number of failures of a given category to a given unit of measure, e.g., failures per unit of time, failures per number of transactions, failures per number of computer runs.

fault attack

See Also: negative testing

Synonyms: attack

Directed and focused attempt to evaluate the quality, especially reliability, of a test object by attempting to force specific failures to occur.

feature

Ref: After IEEE 1008

Synonyms: software feature

An attribute of a component or system specified or implied by requirements documentation (for example reliability, usability or design constraints).

formal review

A review characterized by documented procedures and requirements, e.g., inspection.

functional requirement

Ref: IEEE 610

A requirement that specifies a function that a component or system must perform.

functional testing

See Also: black-box testing

Testing based on an analysis of the specification of the functionality of a component or system.

functionality

Ref: ISO 9126

The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

high-level test case

See Also: low-level test case

Synonyms: abstract test case , logical test case

A test case without concrete (implementation level) values for input data and expected results. Logical operators are used: instances of the actual values are not yet defined and/or available.

impact analysis

The assessment of change to the layers of development documentation, test documentation and components, in order to implement a given change to specified requirements.

incident

Ref: After IEEE 1008

Synonyms: deviation , software test incident , test incident

Any event occurring that requires investigation.

incident logging

Recording the details of any incident that occurred, e.g., during testing.

incident management

Ref: After IEEE 1044

The process of recognizing, investigating, taking action and disposing of incidents. It involves logging incidents, classifying them and identifying the impact.

incident management tool

See Also: defect management tool

A tool that facilitates the recording and status tracking of incidents. They often have workflow-oriented facilities to track and control the allocation, correction and re-testing of incidents and provide reporting facilities.

incident report

Ref: After IEEE 829

Synonyms: deviation report , software test incident report , test incident report

A document reporting on any event that occurred, e.g., during the testing, which requires investigation.

incremental development model

A development lifecycle where a project is broken into a series of increments, each of which delivers a portion of the functionality in the overall project requirements. The requirements are prioritized and delivered in priority order in the appropriate increment. In some (but not all) versions of this lifecycle model, each subproject follows a mini V-model with its own design, coding and testing phases.

incremental testing

Testing where components or systems are integrated and tested one or some at a time, until all the components or systems are integrated and tested.

independence of testing

Ref: After DO-178b

Separation of responsibilities, which encourages the accomplishment of objective testing.

indicator

Ref: ISO 14598

A measure that can be used to estimate or predict another measure.

informal review

A review not based on a formal (documented) procedure.

input

A variable (whether stored within a component or outside) that is read by a component.

input value

See Also: input

An instance of an input.

inspection

Ref: After IEEE 610, IEEE 1028 **See Also:** peer review

A type of peer review that relies on visual examination of documents to detect defects, e.g., violations of development standards and non-conformance to higher level documentation. The most formal review technique and therefore always based on a documented procedure.

installation guide

Supplied instructions on any suitable media, which guides the installer through the installation process. This may be a manual guide, step-by-step procedure, installation wizard, or any other similar process description.

integration

The process of combining components or systems into larger assemblies.

integration testing

See Also: component integration testing, system integration testing

Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.

interface testing

An integration test type that is concerned with testing the interfaces between components or systems.

interoperability

Ref: After ISO 9126 **See Also:** functionality

The capability of the software product to interact with one or more specified components or systems.

interoperability testing

See Also: functionality testing

Synonyms: compatibility testing

Testing to determine the interoperability of a software product.

invalid testing

See Also: error tolerance, negative testing

Testing using input values that should be rejected by the component or system.

iterative development model

A development lifecycle where a project is broken into a usually large number of iterations. An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows from iteration to iteration to become the final product.

keyword-driven testing

See Also: data-driven testing

Synonyms: action word-driven testing

A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test.

level test plan

See Also: test plan

A test plan that typically addresses one test level.

load testing

See Also: performance testing, stress testing

A type of performance testing conducted to evaluate the behavior of a component or system with increasing load, e.g., numbers of parallel users and/or numbers of transactions, to determine what load can be handled by the component or system.

load testing tool

See Also: performance testing tool

A tool to support load testing whereby it can simulate increasing load, e.g., numbers of concurrent users and/or transactions within a specified time-period.

maintainability

Ref: ISO 9126

The ease with which a software product can be modified to correct defects, modified to meet new requirements, modified to make future maintenance easier, or adapted to a changed environment.

maintainability testing

Synonyms: serviceability testing

Testing to determine the maintainability of a software product.

maintenance

Ref: IEEE 1219

Modification of a software product after delivery to correct defects, to improve performance or other attributes, or to adapt the product to a modified environment.

maintenance testing

Testing the changes to an operational system or the impact of a changed environment to an operational system.

master test plan

See Also: test plan

A test plan that typically addresses multiple test levels.

maturity

Ref: ISO 9126 **See Also:** Capability Maturity Model Integration, Test Maturity Model integration, reliability

(1) The capability of an organization with respect to the effectiveness and efficiency of its processes and work practices. (2) The capability of the software product to avoid failure as a result of defects in the software.

maturity model

A structured collection of elements that describe certain aspects of maturity in an organization, and aid in the definition and understanding of an organization's processes. A maturity model often provides a common language, shared vision and framework for prioritizing improvement actions.

measure

Ref: ISO 14598

The number or category assigned to an attribute of an entity by making a measurement.

measurement

Ref: ISO 14598

The process of assigning a number or category to an entity to describe an attribute of that entity.

memory leak

A memory access failure due to a defect in a program's dynamic store allocation logic that causes it to fail to release memory after it has finished using it, eventually causing the program and/or other concurrent processes to fail due to lack of memory.

metric

Ref: ISO 14598

A measurement scale and the method used for measurement.

milestone

A point in time in a project at which defined (intermediate) deliverables and results should be ready.

modeling tool

Ref: Graham .

A tool that supports the creation, amendment and verification of models of the software or system.

moderator

Synonyms: inspection leader

The leader and main person responsible for an inspection or other review process.

monitoring tool

Ref: After IEEE 610.

A software tool or hardware device that runs concurrently with the component or system under test and supervises, records and/or analyses the behavior of the component or system.

multiple condition coverage

Synonyms: branch condition combination coverage , condition combination coverage

The percentage of combinations of all single condition outcomes within one statement that have been exercised by a test suite. 100% multiple condition coverage implies 100% modified condition decision coverage.

non-functional requirement

A requirement that does not relate to functionality, but to attributes such as reliability, efficiency, usability, maintainability and portability.

non-functional testing

Testing the attributes of a component or system that do not relate to functionality, e.g., reliability, efficiency, usability, maintainability and portability.

open source tool

A software tool that is available to all potential users in source code form, usually via the internet. Its users are permitted, usually under license, to study, change, improve and, at times, to distribute the software.

operational acceptance testing

See Also: operational testing

Synonyms: production acceptance testing

Operational testing in the acceptance test phase, typically performed in a (simulated) operational environment by operations and/or systems administration staff focusing on operational aspects, e.g., recoverability, resource-behavior, installability and technical compliance.

operational profile

The representation of a distinct set of tasks performed by the component or system, possibly based on user behavior when interacting with the component or system, and their probabilities of occurrence. A task is logical rather than physical and can be executed over several machines or be executed in non-contiguous time segments.

operational testing

Ref: IEEE 610

Testing conducted to evaluate a component or system in its operational environment.

output

A variable (whether stored within a component or outside) that is written by a component.

pair programming

A software development approach whereby lines of code (production and/or test) of a component are written by two programmers sitting at a single computer. This implicitly means ongoing real-time code reviews are performed.

pass

Synonyms: test pass

A test is deemed to pass if its actual result matches its expected result.

path

Synonyms: control flow path

A sequence of events, e.g., executable statements, of a component or system from an entry point to an exit point.

peer review

A review of a software work product by colleagues of the producer of the product for the purpose of identifying defects and improvements. Examples are inspection, technical review and walkthrough.

performance

Ref: After IEEE 610 **See Also:** efficiency

Synonyms: time behavior

The degree to which a system or component accomplishes its designated functions within given constraints regarding processing time and throughput rate.

performance profiling

The task of analyzing, e.g., identifying performance bottlenecks based on generated metrics, and tuning the performance of a software component or system using tools.

performance testing

See Also: efficiency testing

Testing to determine the performance of a software product.

performance testing tool

A tool to support performance testing that usually has two main facilities: load generation and test transaction measurement. Load generation can simulate either multiple users or high volumes of input data. During execution, response time measurements are taken from selected transactions and these are logged. Performance testing tools normally provide reports based on test logs and graphs of load against response times.

portability

Ref: ISO 9126

The ease with which the software product can be transferred from one hardware or software environment to another.

portability testing

Synonyms: configuration testing

Testing to determine the portability of a software product.

post-execution comparison

Comparison of actual and expected results, performed after the software has finished running.

postcondition

Environmental and state conditions that must be fulfilled after the execution of a test or test procedure.

precondition

Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure.

priority

The level of (business) importance assigned to an item, e.g., defect.

probe effect

The effect on the component or system by the measurement instrument when the component or system is being measured, e.g., by a performance testing tool or monitor. For example performance may be slightly worse when performance testing tools are being used.

process

Ref: ISO 12207

A set of interrelated activities, which transform inputs into outputs.

process cycle test

Ref: TMap **See Also:** procedure testing

A black-box test design technique in which test cases are designed to execute business procedures and processes.

process improvement

Ref: CMMI

A program of activities designed to improve the performance and maturity of the organization's processes, and the result of such a program.

product risk

See Also: risk

A risk directly related to the test object.

product-based quality

Ref: After Garvin **See Also:** manufacturing-based quality, quality attribute, transcendent-based quality, user-based quality, value-based quality

A view of quality, wherein quality is based on a well-defined set of quality attributes. These attributes must be measured in an objective and quantitative way. Differences in the quality of products of the same type can be traced back to the way the specific quality attributes have been implemented.

project

Ref: ISO 9000

A project is a unique set of coordinated and controlled activities with start and finish dates undertaken to achieve an objective conforming to specific requirements, including the constraints of time, cost and resources.

project risk

See Also: risk

A risk related to management and control of the (test) project, e.g., lack of staffing, strict deadlines, changing requirements, etc.

qualification

Ref: ISO 9000

The process of demonstrating the ability to fulfill specified requirements. Note the term "qualified" is used to designate the corresponding status.

quality

Ref: After IEEE 610

The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

quality assurance

Ref: ISO 9000

Part of quality management focused on providing confidence that quality requirements will be fulfilled.

quality attribute

Ref: IEEE 610

Synonyms: quality characteristic , software product characteristic , software quality characteristic

A feature or characteristic that affects an item's quality.

Rational Unified Process (RUP)

A proprietary adaptable iterative software development process framework consisting of four project lifecycle phases: inception, elaboration, construction and transition.

regression testing

Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

reliability

Ref: ISO 9126

The ability of the software product to perform its required functions under stated conditions for a specified period of time, or for a specified number of operations.

reliability growth model

A model that shows the growth in reliability over time during continuous testing of a component or system as a result of the removal of defects that result in reliability failures.

reliability testing

Testing to determine the reliability of a software product.

requirement

Ref: After IEEE 610

A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

requirements management tool

A tool that supports the recording of requirements, requirements attributes (e.g., priority, knowledge responsible) and annotation, and facilitates traceability through layers of requirements and requirements change management. Some requirements management tools also provide facilities for static analysis, such as consistency checking and violations to pre-defined requirements rules.

requirements phase

Ref: IEEE 610

The period of time in the software lifecycle during which the requirements for a software product are defined and documented.

requirements-based testing

An approach to testing in which test cases are designed based on test objectives and test conditions derived from requirements, e.g., tests that exercise specific functions or probe non-functional attributes such as reliability or usability.

result

See Also: actual result, expected result

Synonyms: outcome , test outcome , test result

The consequence/outcome of the execution of a test. It includes outputs to screens, changes to data, reports, and communication messages sent out.

resumption requirements

Ref: After IEEE 829

The defined set of testing activities that must be repeated when testing is re-started after a suspension.

review

Ref: After IEEE 1028

An evaluation of a product or project status to ascertain discrepancies from planned results and to recommend improvements. Examples include management review, informal review, technical review, inspection, and walkthrough.

review tool

A tool that provides support to the review process. Typical features include review planning and tracking support, communication support, collaborative reviews and a repository for collecting and reporting of metrics.

reviewer

Synonyms: checker , inspector

The person involved in the review that identifies and describes anomalies in the product or project under review. Reviewers can be chosen to represent different viewpoints and roles in the review process.

risk

A factor that could result in future negative consequences.

risk analysis

The process of assessing identified project or product risks to determine their level of risk, typically by estimating their impact and probability of occurrence (likelihood).

risk assessment

See Also: product risk, project risk, risk, risk impact, risk level, risk likelihood

The process of identifying and subsequently analyzing the identified project or product risk to determine its level of risk, typically by assigning likelihood and impact ratings.

risk level

The importance of a risk as defined by its characteristics impact and likelihood. The level of risk can be used to determine the intensity of testing to be performed. A risk level can be expressed either qualitatively (e.g., high, medium, low) or quantitatively.

risk likelihood

Synonyms: likelihood

The estimated probability that a risk will become an actual outcome or event.

risk management

Systematic application of procedures and practices to the tasks of identifying, analyzing, prioritizing, and controlling risk.

risk mitigation

Synonyms: risk control

The process through which decisions are reached and protective measures are implemented for reducing risks to, or maintaining risks within, specified levels.

risk-based testing

An approach to testing to reduce the level of product risks and inform stakeholders of their status, starting in the initial stages of a project. It involves the identification of product risks and the use of risk levels to guide the test process.

robustness

Ref: IEEE 610 **See Also:** error-tolerance, fault-tolerance

The degree to which a component or system can function correctly in the presence of invalid inputs or stressful environmental conditions.

robustness testing

Testing to determine the robustness of the software product.

root cause

Ref: CMMI

A source of a defect such that if it is removed, the occurrence of the defect type is decreased or removed.

safety

Ref: ISO 9126

The capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.

scribe

Synonyms: recorder

The person who records each defect mentioned and any suggestions for process improvement during a review meeting, on a logging form. The scribe should ensure that the logging form is readable and understandable.

scripting language

A programming language in which executable test scripts are written, used by a test execution tool (e.g., a capture/playback tool).

security

Ref: ISO 9126 **See Also:** functionality

Attributes of software products that bear on its ability to prevent unauthorized access, whether accidental or deliberate, to programs and data.

security testing

See Also: functionality testing

Testing to determine the security of the software product.

security testing tool

A tool that provides support for testing security characteristics and vulnerabilities.

security tool

A tool that supports operational security.

severity

Ref: After IEEE 610

The degree of impact that a defect has on the development or operation of a component or system.

simulation

Ref: ISO 2382/1

The representation of selected behavioral characteristics of one physical or abstract system by another system.

simulator

Ref: After IEEE 610, DO178b **See Also:** emulator

A device, computer program or system used during testing, which behaves or operates like a given system when provided with a set of controlled inputs.

site acceptance testing

Acceptance testing by users/customers at their site, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes, normally including hardware as well as software.

software

Ref: IEEE 610

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

software integrity level

The degree to which software complies or must comply with a set of stakeholder-selected software and/or software-based system characteristics (e.g., software complexity, risk assessment, safety level, security level, desired performance, reliability or cost) which are defined to reflect the importance of the software to its stakeholders.

software lifecycle

The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software lifecycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. Note these phases may overlap or be performed iteratively.

specification

Ref: After IEEE 610

A document that specifies, ideally in a complete, precise and verifiable manner, the requirements, design, behavior, or other characteristics of a component or system, and, often, the procedures for determining whether these provisions have been satisfied.

stability

Ref: ISO 9126 **See Also:** maintainability

The capability of the software product to avoid unexpected effects from modifications in the software.

standard

Ref: After CMMI

Formal, possibly mandatory, set of requirements developed and used to prescribe consistent approaches to the way of working or to provide guidelines (e.g., ISO/IEC standards, IEEE standards, and organizational standards).

standard-compliant testing

See Also: process-compliant testing

Testing that complies to a set of requirements defined by a standard, e.g., an industry testing standard or a standard for testing safety-critical systems.

state table

A grid showing the resulting transitions for each state combined with each possible event, showing both valid and invalid transitions.

state transition

A transition between two states of a component or system.

state transition testing

See Also: N-switch testing

Synonyms: finite state testing

A black-box test design technique in which test cases are designed to execute valid and invalid state transitions.

statement

Synonyms: source statement

An entity in a programming language, which is typically the smallest indivisible unit of execution.

statement coverage

The percentage of executable statements that have been exercised by a test suite.

statement testing

A white-box test design technique in which test cases are designed to execute statements.

static analysis

Analysis of software development artifacts, e.g., requirements or code, carried out without execution of these software development artifacts. Static analysis is usually carried out by means of a supporting tool.

static analyzer

Synonyms: analyzer , static analysis tool

A tool that carries out static analysis.

static code analysis

Analysis of source code carried out without execution of that software.

static testing

Testing of a software development artifact, e.g., requirements, design or code, without execution of these artifacts, e.g., reviews or static analysis.

stress testing

Ref: After IEEE 610 **See Also:** performance testing, load testing

A type of performance testing conducted to evaluate a system or component at or beyond the limits of its anticipated or specified workloads, or with reduced availability of resources such as access to memory or servers.

stress testing tool

A tool that supports stress testing.

structural coverage

Coverage measures based on the internal structure of a component or system.

stub

Ref: After IEEE 610

A skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component.

system

Ref: IEEE 610

A collection of components organized to accomplish a specific function or set of functions.

system integration testing

Testing the integration of systems and packages; testing interfaces to external organizations (e.g., Electronic Data Interchange, Internet).

system of systems

Multiple heterogeneous, distributed systems that are embedded in networks at multiple levels and in multiple interconnected domains, addressing large-scale inter-disciplinary common problems and purposes, usually without a common management structure.

system testing

Ref: Hetzel

Testing an integrated system to verify that it meets specified requirements.

system under test (SUT)

See test object.

Systematic Test and Evaluation Process (STEP)

See Also: content-based model

A structured testing methodology, also used as a content-based model for improving the testing process. Systematic Test and Evaluation Process (STEP) does not require that improvements occur in a specific order.

technical review

Ref: Gilb and Graham, IEEE 1028 **See Also:** peer review

A peer group discussion activity that focuses on achieving consensus on the technical approach to be taken.

test

Ref: IEEE 829

A set of one or more test cases.

test analysis

The process of analyzing the test basis and defining test objectives.

test approach

The implementation of the test strategy for a specific project. It typically includes the decisions made that follow based on the (test) project's goal and the risk assessment carried out, starting points regarding the test process, the test design techniques to be applied, exit criteria and test types to be performed.

test automation

The use of software to perform or support test activities, e.g., test management, test design, test execution and results checking.

test basis

Ref: After TMap

All documents from which the requirements of a component or system can be inferred. The documentation on which the test cases are based. If a document can be amended only by way of formal amendment procedure, then the test basis is called a frozen test basis.

test case

Ref: After IEEE 610

A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

test case specification

Ref: After IEEE 829 **See Also:** test specification

A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item.

test charter

See Also: exploratory testing

Synonyms: charter

A statement of test objectives, and possibly test ideas about how to test. Test charters are used in exploratory testing.

test closure

See Also: test process

During the test closure phase of a test process data is collected from completed activities to consolidate experience, testware, facts and numbers. The test closure phase consists of finalizing and archiving the testware and evaluating the test process, including preparation of a test evaluation report.

test comparator

Synonyms: comparator

A test tool to perform automated test comparison of actual results with expected results.

test condition

Synonyms: test requirement , test situation

An item or event of a component or system that could be verified by one or more test cases, e.g., a function, transaction, feature, quality attribute, or structural element.

test control

See Also: test management

A test management task that deals with developing and applying a set of corrective actions to get a test project on track when monitoring shows a deviation from what was planned.

test data

Data that exists (for example, in a database) before a test is executed, and that affects or is affected by the component or system under test.

test data preparation tool

Synonyms: test generator

A type of test tool that enables data to be selected from existing databases or created, generated, manipulated and edited for use in testing.

test design

See Also: test design specification

The process of transforming general test objectives into tangible test conditions and test cases.

test design specification

Ref: After IEEE 829 **See Also:** test specification

A document specifying the test conditions (coverage items) for a test item, the detailed test approach and identifying the associated high-level test cases.

test design technique

Synonyms: test case design technique , test specification technique , test technique

Procedure used to derive and/or select test cases.

test design tool

A tool that supports the test design activity by generating test inputs from a specification that may be held in a CASE tool repository, e.g., requirements management tool, from specified test conditions held in the tool itself, or from code.

test environment

Ref: After IEEE 610

Synonyms: test bed , test rig

An environment containing hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test.

test estimation

The calculated approximation of a result related to various aspects of testing (e.g., effort spent, completion date, costs involved, number of test cases, etc.) which is usable even if input data may be incomplete, uncertain, or noisy.

test execution

The process of running a test on the component or system under test, producing actual result(s).

test execution schedule

A scheme for the execution of test procedures. Note: The test procedures are included in the test execution schedule in their context and in the order in which they are to be executed.

test execution tool

A type of test tool that is able to execute other software using an automated test script, e.g., capture/playback.

test harness

A test environment comprised of stubs and drivers needed to execute a test.

test implementation

The process of developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts.

test infrastructure

The organizational artifacts needed to perform testing, consisting of test environments, test tools, office environment and procedures.

test input

The data received from an external source by the test object during test execution. The external source can be hardware, software or human.

test item

See Also: test object

The individual element to be tested. There usually is one test object and many test items.

test level

Ref: After TMap

Synonyms: test stage

A group of test activities that are organized and managed together. A test level is linked to the responsibilities in a project. Examples of test levels are component test, integration test, system test and acceptance test.

test log

Ref: IEEE 829

Synonyms: test record , test run log

A chronological record of relevant details about the execution of tests.

test logging

Synonyms: test recording

The process of recording information about tests executed into a test log.

test management

The planning, estimating, monitoring and control of test activities, typically carried out by a test manager.

test management tool

A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, the logging of results, progress tracking, incident management and test reporting.

test manager

Synonyms: test leader

The person responsible for project management of testing activities and resources, and evaluation of a test object. The individual who directs, controls, administers, plans and regulates the evaluation of a test object.

test monitoring

See Also: test management

A test management task that deals with the activities related to periodically checking the status of a test project. Reports are prepared that compare the actuals to that which was planned.

test object

See Also: test item

Synonyms: system under test

The component or system to be tested.

test objective

A reason or purpose for designing and executing a test.

test oracle

Ref: After Adrion

Synonyms: oracle

A source to determine expected results to compare with the actual result of the software under test. An oracle may be the existing system (for a benchmark), other software, a user manual, or an individual's specialized knowledge, but should not be the code.

test phase

Ref: After Gerrard

A distinct set of test activities collected into a manageable phase of a project, e.g., the execution activities of a test level.

test plan

Ref: After IEEE 829

A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.

test planning

The activity of establishing or updating a test plan.

test policy

A high-level document describing the principles, approach and major objectives of the organization regarding testing.

test procedure specification

Ref: After IEEE 829 **See Also:** test specification

Synonyms: test procedure , test scenario

A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script.

test process

The fundamental test process comprises test planning and control, test analysis and design, test implementation and execution, evaluating exit criteria and reporting, and test closure activities.

test process group (TPG)

Ref: After CMMI

A collection of (test) specialists who facilitate the definition, maintenance, and improvement of the test processes used by an organization.

test progress report

Synonyms: test report

A document summarizing testing activities and results, produced at regular intervals, to report progress of testing activities against a baseline (such as the original test plan) and to communicate risks and alternatives requiring a decision to management.

test reporting

See Also: test process

Collecting and analyzing data from testing activities and subsequently consolidating the data in a report to inform stakeholders.

test run

Execution of a test on a specific version of the test object.

test schedule

A list of activities, tasks or events of the test process, identifying their intended start and finish dates and/or times, and interdependencies.

test script

Commonly used to refer to a test procedure specification, especially an automated one.

test session

See Also: exploratory testing

An uninterrupted period of time spent in executing tests. In exploratory testing, each test session is focused on a charter, but testers can also explore new opportunities or issues during a session. The tester creates and executes on the fly and records their progress.

test specification

A document that consists of a test design specification, test case specification and/or test procedure specification.

test strategy

A high-level description of the test levels to be performed and the testing within those levels for an organization or programme (one or more projects).

test suite

Synonyms: test case suite , test set

A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one.

test summary report

Ref: After IEEE 829

Synonyms: test report

A document summarizing testing activities and results. It also contains an evaluation of the corresponding test items against exit criteria.

test tool

Ref: TMap **See Also:** CAST

A software product that supports one or more test activities, such as planning and control, specification, building initial files and data, test execution and test analysis.

test type

Ref: After TMap

A group of test activities aimed at testing a component or system focused on a specific test objective, i.e. functional test, usability test, regression test etc. A test type may take place on one or more test levels or test phases.

test-driven development (TDD)

A way of developing software where the test cases are developed, and often automated, before the software is developed to run those test cases.

testability

Ref: ISO 9126 **See Also:** maintainability

The capability of the software product to enable modified software to be tested.

tester

A skilled professional who is involved in the testing of a component or system.

testing

The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

testware

Ref: After Fewster and Graham

Artifacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.

traceability

See Also: horizontal traceability, vertical traceability

The ability to identify related items in documentation and software, such as requirements with associated tests.

unit test framework

Ref: Graham

A tool that provides an environment for unit or component testing in which a component can be tested in isolation or with suitable stubs and drivers. It also provides other support for the developer, such as debugging capabilities.

unreachable code

Synonyms: dead code

Code that cannot be reached and therefore is impossible to execute.

usability

Ref: ISO 9126

The capability of the software to be understood, learned, used and attractive to the user when used under specified conditions.

usability testing

Ref: After ISO 9126

Testing to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users under specified conditions.

use case

A sequence of transactions in a dialogue between an actor and a component or system with a tangible result, where an actor can be a user or anything that can exchange information with the system.

use case testing

Synonyms: scenario testing , user scenario testing

A black-box test design technique in which test cases are designed to execute scenarios of use cases.

user acceptance testing

See Also: acceptance testing

Acceptance testing carried out by future users in a (simulated) operational environment focusing on user requirements and needs.

V-model

A framework to describe the software development lifecycle activities from requirements specification to maintenance. The V-model illustrates how testing activities can be integrated into each phase of the software development lifecycle.

validation

Ref: ISO 9000

Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

variable

An element of storage in a computer that is accessible by a software program by referring to it by a name.

verification

Ref: ISO 9000

Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

walkthrough

Ref: Freedman and Weinberg, IEEE 1028 **See Also:** peer review

Synonyms: structured walkthrough

A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content.

white-box test design technique

Synonyms: structural test design technique , structure-based test design technique , structure-based technique , white-box technique

Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

white-box testing

Synonyms: clear-box testing , code-based testing , glass-box testing , logic-coverage testing , logic-driven testing , structural testing , structure-based testing

Testing based on an analysis of the internal structure of the component or system.